Tampere University

Miika Suomalainen

DESIGN AND IMPLEMENTATION OF AN API FOR LOW AND HIGH LEVEL SOFTWARE INTEGRATION OF AN INDUSTRIAL MANIPULATOR

Engineering and Natural Science
Master of Science Thesis
November 2019

ABSTRACT

Miika Suomalainen: Design and implementation of an API for low and high level software integration of an industrial manipulator.
Master of Science thesis
Tampere University
Automation Engineering
November 2019

From the first automated manufacturing line, the main aim of automation and robotics has been to increase work efficiency. Over the years many other sub-goals like improving safety, reducing mistakes and freeing up human labour to more important tasks, have increased the value of robots in automation projects. For decades, the automated manufacturing benefitted most the large companies with products created in large quantities with minimal alteration in products. The small and medium companies could not harvest the benefits of automation because automation and industrial robot systems were huge investments that took a lot time and effort to build and alter. This trend has faced astounding shift during the 21st century.

Today's factory-robot solution development has inherited many design techniques from software world. Modularity, flexibility and easy implementation have become more desirable design goals in automation and manufacturing solutions. Fast commissioning minimize automation line downtime. Modularity and flexibility make solution more scalable. One popular way to achieve modularity is wrapping the functionality using web service architecture. With this architecture, the robot services could be described, published and invoked over a network.

Even though software design architecture is a create tool to accomplish modularity, the robot systems are so versatile that there is no possibility to design a system that could adapt to every situation. Every industry has different demands concerning to factory robot solutions. Therefore, to minimize development time the methodological approach is needed. Methodology helps meeting the constraints caused by humans and physical environment. A system development methodology is a plan that is made to form and control the process of developing. This methodology is necessary because it is not possible to move automation project forward without prior work.

This thesis present a methodology for industrial manipulator cell development from hardware decision and installation to software development solutions. The hardware decisions of methodology will define development guidelines for robot placement in cell, robot connection wiring and external module connectivity. Software section defines development phases for robots internal software development, module software development and development of possible APIs.

Finally, the methodology will be used in practice to develop and implement an industrial manipulator system in Factory Automation Systems and Technologies Laboratory. The implementation was successfully delivered to 12-cell assembly line. The solution demonstrates an efficient approach for factory-robot system design, implementation and initialization offering modularity, flexibility and fast initialization.

Keywords: factory robot, methodology, modularity, API, web service.

TIIVISTELMÄ

Miika Suomalainen: Matalan ja korkean tason ohjelmointirajapintojen suunnittelu ja toteutus teollisuusmanipulaattorisovelluksessa.
Diplomityö
Tampereen yliopisto
Automaatiotekniikka
Marraskuu 2019

Ensimmäisistä automatisoiduista tuotantolinjoista lähtien automatiikan ja robotiikan tavoitteena on ollut korkeampi työskentelytehokkuus. Vuosien varrella tavoitteiden kirjo on laajentunut koskemaan työskentelytuvallisuutta, virheellisten tuotteiden minimointia ja ihmistyövoiman vapauttamista mihin tärkeämpiin tehtäviin. Vuosien ajan automatisoitu tuotanto hyödytti pääasiassa suuria yrityksiä, jotka tuottivat suuria määriä tietyn tyyppisiä tuotteita. Pienet ja keskisuuret yritykset eivät päässeet hyödyntämään automaation ja robotisoinnin mukanaan tuomia etuja yhtä tehokkaasti sen joustamattomuuden ja suuren käyttöönottoinvestoinnin vuoksi. tämä kehityssuunta on kuitenkin muuttunut radikaalisti 2000-luvun aikana.

Tämän päivän teollisuusrobottiratkaisujen kehitystyö on perinyt useita suunnittelukäytäntöjä ohjelmistokehityksen maailmasta. Modulaarisuus, joustavuus ja helppo toteutus ovat tulleet yhä tärkeämmiksi suunnittelutavoitteiksi automaatio-, ja tuotantoprojekteissa. Nopea käyttöönotto minimoi automaatiolinjaston odotusajan. Modulaarisuus ja joustavuus tekevät ratkaisusta skaalautuvamman. Yksi suosittu tapa saavuttaa modulaarisuus, on hyödyntää verkkopalvelu arkkitehtuuria. Arkkitehtuurin avulla robotin tarjoamat palvelut voidaan kuvata, julkistaa ja herättää verkon välityksellä.

Vaikka verkkopalvelu arkkitehtuuri on mainio työkalu modulaarisuuden saavuttamiseksi, robottijärjestelmät ovat niin monimuotoisia, että ei ole mahdollista suunnitella ratkaisua, joka taipuisi jokaiseen käyttötarkoitukseen. Jokaisella teollisuudenalalla on omat vaateensa ja rajoitteensa koskien tuotantolinjastoja. Tämä luo tarpeen metodologialle, jonka avulla kehitystyötä saadaan nopeammaksi. Järjestelmän kehitys metodologia on suunnitelma, joka auttaa muodostamaan, suunnittelemaan ja kontrolloimaan kehitysprosessia. Metodologia on tarpeellinen, koska ilman pohjatyötä automaatioprojektia olisi mahdollista saattaa luotettavasti loppuun asti.

Tämä työ esittelee metodologian teollisuusrobotin työsolun kehitykselle. Metodologia käsittelee laitteistovalintoja sekä asennusratkaisuita, sekä ohjelmistokehitysratkaisuita. Metodologian laitteistopäätökset määrittelevät kehitysraamit robotin sijoittamiselle työsoluun, robotin liitäntöjen sekä ulkoisien moduulien yhteyksien suunnittelulle. Ohjelmisto-osio määrittelee kehitysvaiheet robotin sisäiselle ohjelmistolle, ulkoisien moduulien ohjelmistolle sekä mahdollisesti luotavien rajapintojen suunnittelulle.

Lopuksi metodologia suoritetaan käytännössä teollisuusrobotin kehitys- sekä toteutustyössä. Työ suoritetaan Tampereen yliopiston teollisuusautomaation testilaboratoriossa. Toteutus suoritetaan kaksitoistasoluiselle tuotantolinjastolle. Ratkaisu esittelee tehokkaan lähestymistavan teollisuusrobottisysteemin suunnittelemiselle, toteutukselle ja käyttöönotolle. Toteutuksessa painotetaan modulaarisuutta, joustavuutta ja nopeaa käyttöönottoa.

Avainsanat: Teollisuusrobotti, metodologia, modulaarisuus, rajapinta, verkkopalvelu.

PREFACE

This Master Thesis is a result of almost eight-year long journey among various technical topics. It was fulfilled out in the Department of Production Engineering in the FAST laboratory.

I want to express my deepest gratitude to Professor José L. Martínez Lastra for offering the opportunity to work among this practical project.

I would like to thank my thesis supervisor Luis E. Gonzalez Moctezuma. I truly appreciate the professional guidance he has given me throughout this research project.

Words cannot describe how grateful I am for the members of my family. I would like to say thank you to Merja Suomalainen, Ari Suomalainen, Sami Suomalainen and Aila Suomalainen who all had their unique way to push me to where I am today.

I also want Sonja Salomaa to know how much I value the support she has offered me during this process. She knew just the right words when I was lacking motivation. Thank you from the bottom of my heart.

I am eternally grateful for my former math teacher Timo Ojansivu. The enthusiasm he has towards teaching little minds had huge effect for my motivation to pursue knowledge. His passion for math sealed my decision to pursue career among technology.

Finally yet importantly, I would like to thank Oskari Kankare, Annimari Hartikainen, Petra Hanhijoki, Teemu Laine, Mikko Immonen, Mikael Silenius, Tomi Korvela, Miska Lehtinen, Kristian Klemets, Ilmari Kyyrönen, Riku Ahtiala and Janne Salminen. They have spent countless hours in class with me. They have written countless of exercises with me. They have drank countless of beers with me. They made this journey just as amazing as it was.

Tampere, 29 November 2019

Miika Suomalainen

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ACE	Automation Control Environment
AM	Additive Manufacturing
API	Application Programming Interface
CAD	Computer Aided Design
CSS	Cascading Style Sheet
HLP	High Level Programming
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IP	Internet Protocol
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
MES	Manufacturing Execution System
OSI	Open Systems Interconnection
PNG	Portable Network Graphics
REST	Representational state transfer
RTU	Remote Terminal Unit
SCARA	Selective Compliant Assembly Robot Arm
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
STL	Stereolithography
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
UDDI	Universal Description Discovery and Integration
UML	Unified Modeling Language
WSDL	Web Service Description Language
XML	Extensible Markup Language

1. INTRODUCTION

1.1 Background

Automation in the manufacturing industry started with use of simple pneumatic and hydraulic systems. Industrial operations are usually automated to boost production and minimizing the cost of labour. Industrial automation has achieved huge improvements among tasks that were previously executed manually. The results of using latest technologies to automate existing processes typically lead to improved efficiency, reduced labour costs and higher quality in products.

The first assembly line was introduced in 1913 by the Ford Motor Company. From that time, the inventions in the supportive fields of automation and robotics have increased the efficiency and capability of manufacturing industry. The advances of technologies like mechatronics, computing and wireless communication have pioneered a fast growing field of robotics. This growth offers an opportunity for a huge number of real time applications. Today's industrial robots feature vision camera systems, improved operational degrees of freedom and high computing speed. However, to operate correctly the environment of these robots need to be highly structured. Robot industry also suffers of inflexibility caused by highly specialized products. The complex systems formed by these robots are usually highly distributed and include numerous hardware and software modules. Traditionally these sensors, actuators and controllers cooperate to achieve specified tasks. This way of designing systems to achieve specific tasks and to be manufactured as one unit has slowly changed towards the modular way of thinking. This is because integration of robots into existing facilities can be difficult and expensive.

Today's robot systems are usually created with modular design and implementation. Robots are comprised of multiple different hardware components that communicate with each other. The software in these hardware components are often developed with different programming languages and different communication mechanisms. To simplify the future development process in factories, the design needs to be modular, interoperable and easily configurable. This speeds up development process, ease later scaling and therefore increases efficiency and production time.

Modular design is an approach that divides a system into smaller subsystems. These subsystems are called modules. Once these modules are created, they can be used in different system assemblies. Standard interfaces between modules are crucial to successful modular design. This approach can be applied in software and hardware level, which brings many challenges with it. To apply modular design successfully developers need to practice system analysis and understand the constraints the hardware brings with it. This includes selection of communication interface, protocol and architecture.

1.2 Problem definition

Automated manufacturing line is a multimillion tool that may over time let to be too effective or too ineffective for the job it was originally constructed. This all is a result in constantly changing market conditions worldwide. When automated production line does not serve the need as well as it should, the company needs to make decisions between building a new production line or modify the old one. Either way, if modular design is applied, the costs of investment are going to decrease significantly.

The supplier of the automated manufacturing system faces the challenge of changing market conditions as well. To maintain existing customers in changing conditions, the supplier needs to offer modular, easily reconfigurable and flexible systems so that in the case of choosing between building new and modifying existing manufacturing line, the client chooses to modify existing. Supplier gets to keep client and does not need to compete with other suppliers. Problem is to build the system so that possible changes could be carried out with minimal financial loss. To accomplish this the supplier needs to have significant knowledge of the domain client is acting in, design the system to be quickly initialized to minimize downtime and face the challenges of combining varying hard- and software modules with their unique constraints.

Due to complexity of modern automation systems, it is common that supplier obtains some or all hardware components from third party providers. The compatibility of these components is important, but does not alone equal to functional system. In addition, of shared communication protocols, hardware components need successfully interact with each other. This interoperability is usually achieved with software level configuration and programming. In complex systems, this is time consuming and need to be well designed to maintain modular design.

1.3 Objectives and scope

This thesis includes all basic phases of robot system installation, but focus will be on software level designing. The main goal of this case study and research work is to answer following questions:

- How to integrate external modules to robot system maintaining modularity?
- How to identify and design multi-level communicational architecture between compatible hardware modules?
- How to create and transfer free shape path information to robot most efficient and scalable way?

The scope of this thesis will be established around these questions. However this thesis is written as a part of robot commissioning project, some less related topics, like choosing robot location in work cell, will be addressed. These topics offer the environment and constraints to all software design choices.

1.4 Outline

This thesis is divided into five chapters. The outline of these chapters is as follows: Chapter one declares introduction, the problem and the objectives to this case study. Chapter two offers the theoretical background and introduces concepts used to solve problems later in this document. Chapter three includes the documentation of proposed methodology to achieve objectives mentioned in chapter 1. Chapter four introduces the implementation of proposed approach. In last chapter, chapter five, the conclusions will be declared and some suggestion for future work are presented.

2. LITERATURE AND INDUSTRIAL PRACTICE REVIEW

The main goal of this chapter is to explain what is manufacturing system and what are the essential components within it. First, there will be introduction about what are industrial robots, how they can be used in the industrial field and what different methods there are to program these robots. Moreover there will be briefly explanation of what are TCP/IP-protocol and web service architecture and what are their purpose in the industrial systems. We also explain what additive manufacturing is and what is the process behind creating a physical object using 3D printing. Lastly, there will be an explanation of what are XML-based (Extensible Markup Language) vector image format called Scalable Vector Graphics and how free-form geometry points can be calculated using De Casteljau Algorithm.

2.1 Industrial automation

2.1.1 Automated manufacturing

Literature in the field defines manufacturing system as a system that uses raw materials as input and produces final products as output.[1] Today the amount of materials and products produced is broad and expands every day. There are numerous examples that are processed with manufacturing systems like metals, glass, plastics, chemicals pulp, food, home appliances, automobiles and electronics. Processes used in manufacturing include machining such as drilling, grinding and cutting. Materials are handled using for example conveyors, robotic loaders/unloaders, painting, casting or part assembly.[2] We talk about automated manufacturing, when manufacturing is executed with sequence of preplaced operations with minimized or totally removed human labour. Manufacturing process is in this case controlled and performed using specialized equipment and devices. [3]

Robots are a specific form of automation. They are mainly used in discrete automation systems. Factory or Discrete automation consists of fast execution of occasional movements. This usually involves moving large machine parts with highly dynamic motion. All movements must be executed with great precision. The overall production plant generally involves independently automated manufacturers that consists of numerous machines.[4] Industrial robots have many abilities that make them useful for a manufacturing usage: measurement, manipulation and material handling. Each object need to be relocated from one location in the factory to another in order to be manufactured, stored, packed or assembled. Robot is an ideal contender for material handling operations because of the robots ability to move a picked object in space using predefined paths, without altering the physical characteristics of the object.[5]

2.1.2 Industrial robots

According to the Robotic Industries Association, an industrial robot is an automatically controlled, reprogrammable, multipurpose manipulator that can be programmed in three or more axes which may be either fixed in place or mobile for use in industrial automation applications. [6] Depending on the type of the robot, an industrial robot is a combination of the following parts and components: hand(manipulator), wrist, arm, base, memory, library of programs programmed by the user, safety interlocks, computer interface, power supply and controller.[7; 8]

Robots are often classified by the shape of work envelope that their manipulator is able to reach. Different robot types by this classification are called: Cartesian-coordinates robot, Cylindrical-coordinates-robot, SCARA (Selective Compliant Assembly Robot Arm) robot, polar-coordinates robot and revolute coordinates robot. Revolute-coordinates robots are also called jointed arm robot because its design mimics human arm. [7]

Industrial robots are used in various ways in various fields of industry. With attached welding tool they are able to weld car body components up to 500kg in manufacturing line.[9] Painting industry benefits the precision of industrial robots to create high quality paintjob.[10] And with added sensing and safety equipment robots are capable to work in same working space with humans in human robot collaborative assembly tasks.[11]

2.1.3 Robot programming

When it comes to industrial robot programming, there are two main methods, offline and online programming. Online programming requires the user to actively use the pendant. User needs to move the end-effector manually to the wanted positions and orientations at each step of the robot tasks. These movements are recorded to robot memory and executed as robot program, that commands the robot to move though the pre-recorded tool locations.[12] Teaching trajectories with online programming is very time consuming and needs to be repeated every time there is even a little change in the executed task.[13] Offline programming method makes possible to achieve needed functionality without the limitations of online programming. These systems takes advantage on CAD (Computer Aided Design) model of the robot and the environment robot is placed in. This makes possible to program and simulate the tasks before converting them to a robot program. Offline programming is a programming method that realizes in computer environment. Motion and process simulation could be made without accessing the robot itself. This includes path planning, collision detection and program design.[14] This approach might be problematic if the geometric description of the environment or the work piece cannot be verified. [15]

Certain knowledge of manufacturing systems and robotics is needed when programming the industrial robots. No matter if, programming style is online mode using pendant control device, or offline mode using text-based programming, the programmer need to have strong background in robotics. There have been development to make programming software easy, intuitive and usable without time-consuming learning steps by using

Augmented reality,[16] wearable controllers,[17; 18] or by human-machine demonstration.[19] These high-level programming techniques can overcome the drawbacks of classical approaches since they could ease the programming process for the user. The basic idea with HLP(High Level Programming) systems is to allow humans to teach a task solution to a robot using a human-like procedures like gestures and speech.[20]

2.2 Computer aided manufacturing

2.2.1 Computer-aided design

Computer aided design refers to the integration of computers into the production process to improve productivity. CAD-systems store, retrieve, manipulate and display graphical information.[21] These systems make possible to generate three-dimensional models within the computer and from those models generate drawings for manufacturing purposes. More complex design systems allow analysis of the computer model to take place. For example checking or interference between pars of an assembly, calculation of surface area, and mass properties.[22]

The advantages of computer-aided design are reduced drafting time, easier and faster changes to drawings and increased accuracy and quality.[23] When user does not need to worry about drawing and rendering skills, he or she can concentrate on problem solving, fundamental design and judgement.[24] In economic view, the system of Computer-aided engineering help to reduce the costs for full-scale experiment and prototyping. These systems make it possible to get finished product faster to market.[25]

2.2.2 Additive Manufacturing

Additive manufacturing is referenced in literature as “Layer-by-layer process of producing 3D objects directly from digital model”. Where conventional or subtractive manufacturing processes, such as drilling or milling create a part or product by taking material away from the work-piece, additive manufacturing piles successive layers to build a finished piece.[26]

The generic AM (Additive Manufacturing) process involves multiple steps that shift from the virtual CAD-file to the physical resultant object. Process starts with creating a 3D computer model. This may be done with CAD-software or by scanning an existing object. This 3D model needs to describe the external geometry of the part. After 3D model is created, it needs to be converted to suitable file format. STL- (Stereolithography) file format has become a de facto standard in the field of 3D printing. STL-file include information about the external closed surfaces of the CAD model. STL-file is uploaded to program that converts the description of closed surfaces to actual movements of the manufacturing machine. This program is generally referred as “slicer”. Slicer divides 3D object into numerous thin horizontal cross-sections. These cross-sections form the complete part when stacked again. With slicer the user is able to tune various variables like

layer height, infill percent or heat of the printing nozzle. After all settings are in place, slicer generates g-code file that includes actual command lines to control the printer. This g-code file is then uploaded to 3d-printer witch reads the file line by line to produce the actual physical object. The whole process is described in the figure 1 [27-29]

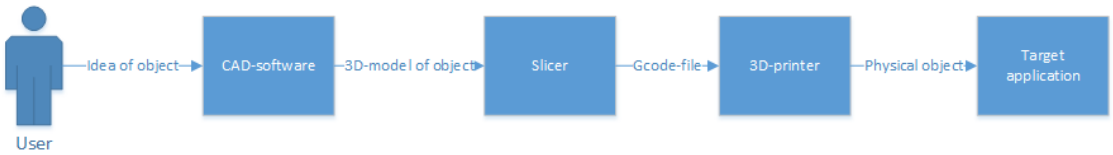


Figure 1. *Additive manufacturing process.*

2.3 Web services and networking

2.3.1 TCP/IP model

Computers are often described as devices or systems that are capable of running a numerous processes. Communication between two devices is made possible by computer networks. This communication is executed by sending and receiving binary information. In order to make sense on each other bits, processes need to agree on a protocol. A protocol is a collection of rules that specify all aspects of data communication.[30] Network model is the system of network protocols. The internet protocol suite is a set of communication protocols that is commonly known as TCP/IP (Transmission Control Protocol/Internet Protocol) because of the two fundamental protocols are Transmission control protocol and internet protocol. [31]

TCP/IP consists of multiple different protocols. To characterize the communication functions of a computing system without regard to its underlying structure and technologies, the Open Systems Interconnection model was developed. OSI-model (Open Systems Interconnection) consists of seven different layers that each have certain characteristics that define it. The TCP/IP stack in terms of the OSI-model is shown in figure 2.[32]

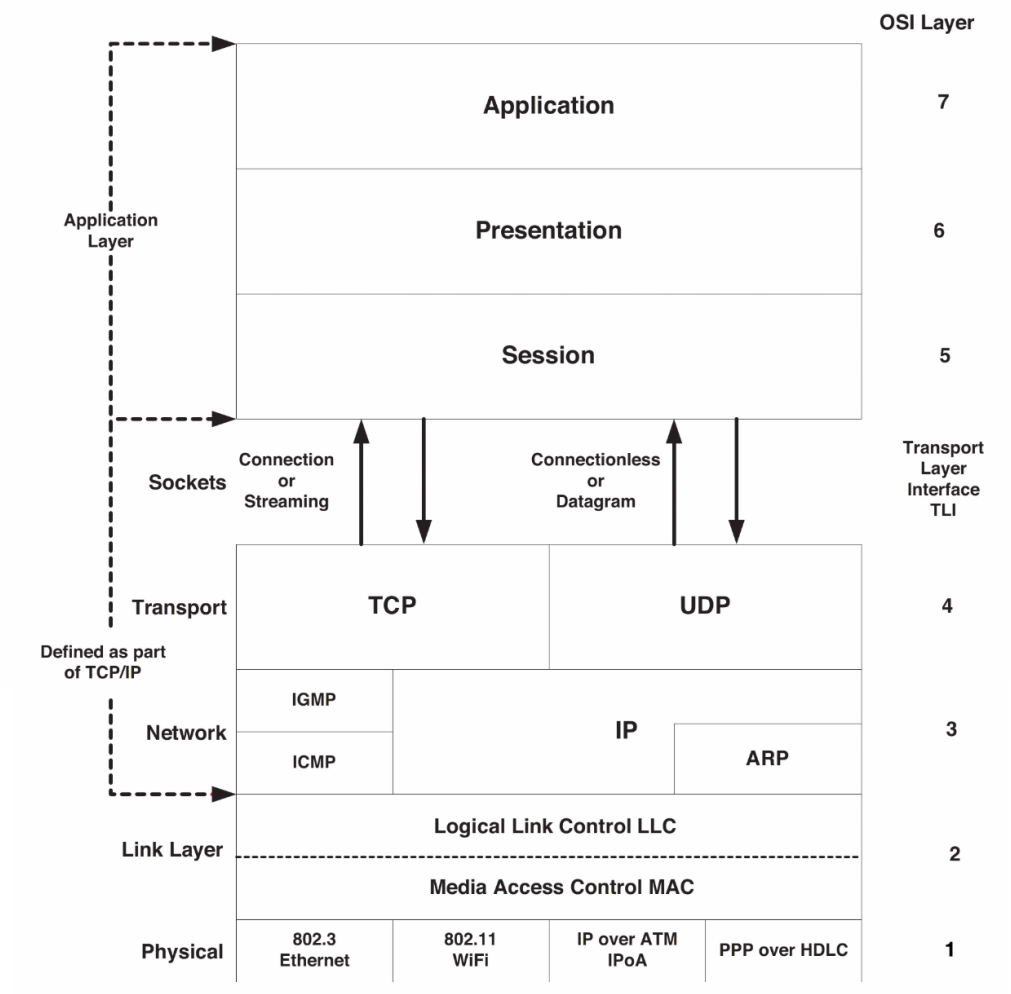


Figure 2. *TCP/IP and the OSI seven-layer model. [32]*

As seen in the figure 2 TCP/IP stack Operates at the four lowest OSI-model layers. Lowest, physical layer is the only touchable infrastructure that supports everything above. It covers all the physical aspects of data-transfer like physical fibre, cobber cables and volts. Layer two, the data link layer includes the switches in the network. It encapsulates bits into frames that are easier to be routed by the network layer. Data link layer also works out how the physical layer is connected. Third layer of the OSI-model is called network layer. It handles all the routing functions and ensures that packets on the transport layer have rational addresses to move to. Fourth layer, transport layer encloses bits into bundles called ‘data packets’. These packets are mainly used to figure out if the data has reached the receiver or not. [33]

2.3.2 Service oriented architecture

Description of a software system in terms of its major components, the relationships between these components and the information that passes through them is called software architecture. The goal is to build systems with well-defined requirements and, systems that have the characteristics needed to meet those requirements now and in the future. An essential function of software architecture is to minimize the complexity of software systems. This also helps managing software in the situations of modifications that systems without exception undergo due to external changes in the business, organizational and technical environments.[34]

Service Oriented Architecture is an architectural paradigm used to enable consumer and service provider interactions via services. These types of services are independent of any product, vendor or technology. Types of services can be combined with other services to provide complete functionality of a large software application. The simplest approach for implementing SOA(Service Oriented Architecture) is to use the Web Services.[35]

Web services are a method of communication between two computing devices over a network. Communication happens in standardized ways using following elements:

- XML- Extensible Markup Language/JSON - JavaScript Object Notation
- WSDL - Web Service Description Language
- SOAP - Simple Object Access Protocol
- UDDI - Universal Description, Discovery and Integration

XML and JSON are the data formats that provides metadata for the data that it contains. SOAP is a messaging protocol specification used in data transfer. WSDL is an XML-based interface description language build to define available services to be consumed. UDDI offers the list of services available. The three main roles in WSA(Web Service Approach) are Service Provider, Service Consumer and Service Broker. The relationships between these roles are presented in figure 3. [36]

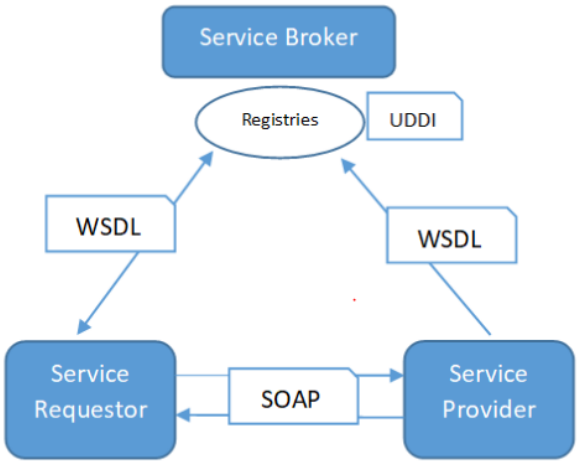


Figure 3. *Roles and relationships of WSA. [36]*

One of the most significant characteristics of Web Services is the fact that data exchange is executed with open standards. After a message is sent from one Web service to another it travels using a Simple Object Access Protocol that relies on application layer protocols like HTTP(Hyper Text Markup Language) of the Internet Protocol Suite. Thanks to open data exchange standards, the platform technology that has been used to create a service does not prevent service-oriented solutions from interoperating. [37]

2.3.3 REST

There are many architectural styles that exploits SOA(Service Oriented Architecture) and Web Service architecture by adding some constraints to architectural styles. One widely used architectural style is called REST(Representational State Transfer). REST itself is not an architecture. It is a set of constraints that can be implicated to design of a system to create a software architectural style. When implemented, we end up with a system that has specific roles for data, components, hyperlink, communication protocols and data consumers. [38] The formal REST constraints are: Stateless, Client-server, Cache, Layered system, Interface, Code-On-Demand. Each constraints is a pre-determined design decision that may have both negative and positive impacts. [39]

Web services are web servers built to support the need of other applications. Client programs use APIs to communicate with web services. An API is used to expose a set of functions and data to maintain interactions between applications and to make information exchange possible. The REST architectural style is commonly applied to the web services APIs design. A REST API consists of an ensemble of linked resources. This resource set is called as the REST API's resource model. [40]

2.4 2D Computer graphics

2.4.1 Scalable vector graphics

SVG(Scalable vector graphics) is a language created by the World Wide Web Consortium (W3C) to manage vector graphic display and animation in XML. SVG is an open, standards-based solution for vector graphics that is entirely text-based so its functions and content are never hidden from users. Due to text-based approach, SVG is editable in text editors and its source code is easily viewable.[41]

Where graphic formats like JPEG(Joint Photographic Express Group) or PNG(Portable Network Graphics) are good for displaying complex images where detail is essential, SVG is superior for showing simple clear line drawings or 2D images. SVG as a format has many benefits when used to display vector images on the web. Images do not loose quality when resizing, they can be animated or manipulated using JavaScript or CSS(Cascading Style Sheets) and they can be printed at any resolution. [42]

2.4.2 De Casteljau Algorithm

The De Casteljau Algorithm is probably the most used one in the field of curve and surface design. Rational Bezier curves are one of the standard representations for free-form geometry in computer aided design. [43] The algorithm is based on simple construction given for the generation of a parabola. The generalization will lead to Bezier curves. Let b_0, b_1, b_2 be any three points in E^3, and let $t \in R$, Construct.

$$b_0^1(t) = (1-t)b_0 + tb_1$$
$$b_1^1(t) = (1-t)b_1 + tb_2$$
$$b_0^2(t) = (1-t)b_0^1(t) + tb_1^1(t)$$

Inserting the first two equations into the third one, results.

$$b_0^2(t) = (1-t)^2 b_0 + 2t(1+t)b_1 + t^2 b_2$$

This represents a quadratic expression in t and so $b_0^2(t)$ traces out a parabola as t varies from $-\infty$ to $+\infty$. This construction consists of repeated linear interpolation. The geometry is illustrated in figure 4. [44]

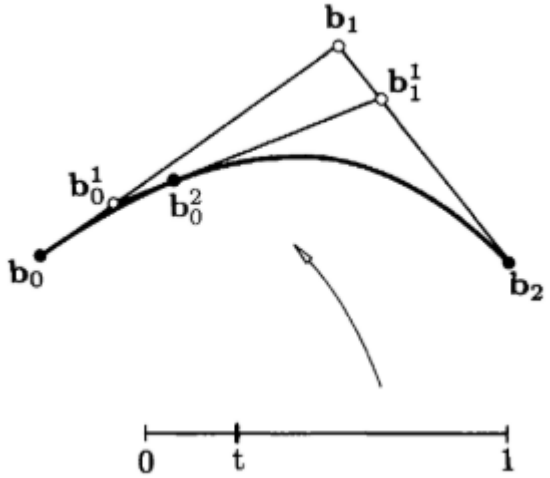


Figure 4. *Construction by repeated linear interpolation.[44]*

Parabolas are plane curves but many applications require true space curves. For those purposes the previous construction for parabola may be generalized to create a polynomial curve of arbitrary degree n. This presentation is also known as de casteljau algorithm.

Given: Bezier points $b_i \; for \; i = 0, \dots, n \; and \; parameter \; t \in [0,1]$.

Find: $The \; point \; b_0^n(t) on \; the \; curve$.

Compute: $Set \; b_i^0 = b_i \; and \; compute \; the \; points$

$$b_i^r(t) = (1-t)b_i^{r-1} + tb_{i+1}^{r-1} \quad \begin{cases} r = 1, \dots, n \\ i = 0, \dots, n-r \end{cases}$$

Figure 5 illustrates the algorithm for cubic at $t = 1/4$. This recursive algorithm consists of repeated linear interpolation. Each step builds a new point from two previous points in the ratio $t: (1-t)$.[45]

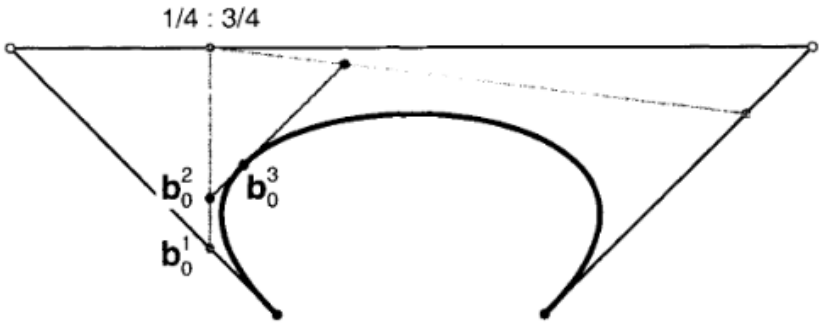


Figure 5. *The de Casteljau algorithm applied to a cubic curve for t = ¼.[45]*

3. METHODOLOGY

In this chapter, a methodology is created to support the development process of service oriented factory robot system design. The goal of this methodology is to offer guidelines for Factory robot hardware decisions and software development. This will speed up the development process and shorten total delivery time. Using the methodology benefits the clients and in addition, the engineer developers, who do not need to invent wheel again every time they start new project. Methodology offers also a way for continuous improving of the design process. Designed methodology is introduced in figure 6. Steps are given more explanatory later in this chapter.

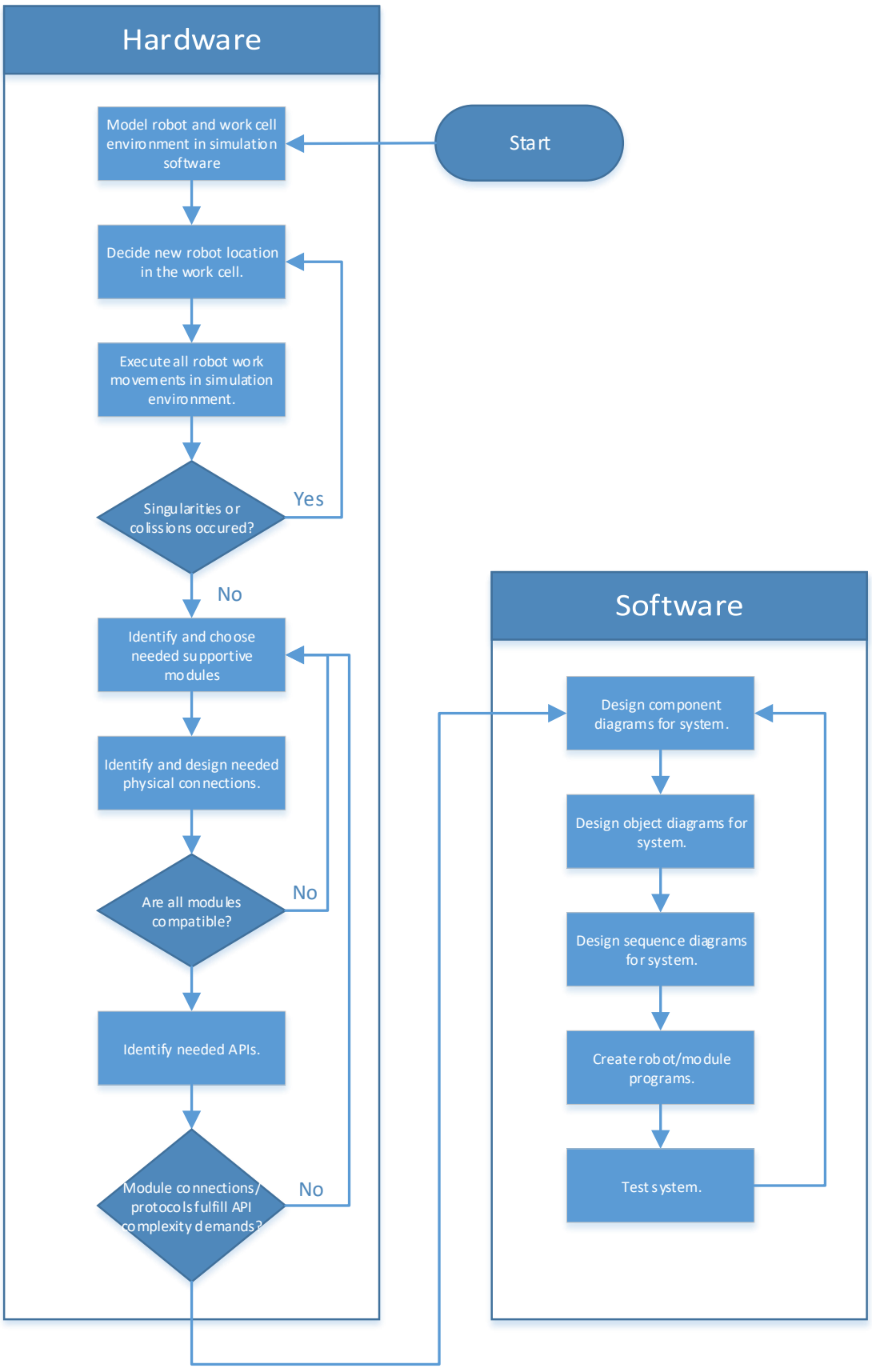


Figure 6. *Methodology for designing service oriented factory robot system.*

Methodology is divided in to two sections. Hardware and software sections. The hardware section deals with physical aspects and connectivity of the robot and additional modules. The objective of hardware section is to end up with ensemble of modules that are capable to execute physical tasks required and communicate at the level that is needed to accomplish the system objectives. Software section address a way to systematically identify needed software components in robot software design. The goal of this section is to encapsulate the system with APIs and provide guidelines for creating software architecture. On each step three main things are considered, What is the context of the step, What are key issues and what are the criteria for successful step accomplishment.

3.1 Hardware design methodology

The methodology starts with designing the hardware component installation, wiring and deciding used connections. Components include factory robot and a set of external modules. External modules may include modems, RTUs(Remote Terminal Units), cameras, robot tool and other modules needed to achieve cell functionality. Web service interface may be executed with robot or external module depending on demands of needed interface. Example of installed hardware components is introduced in figure 7.

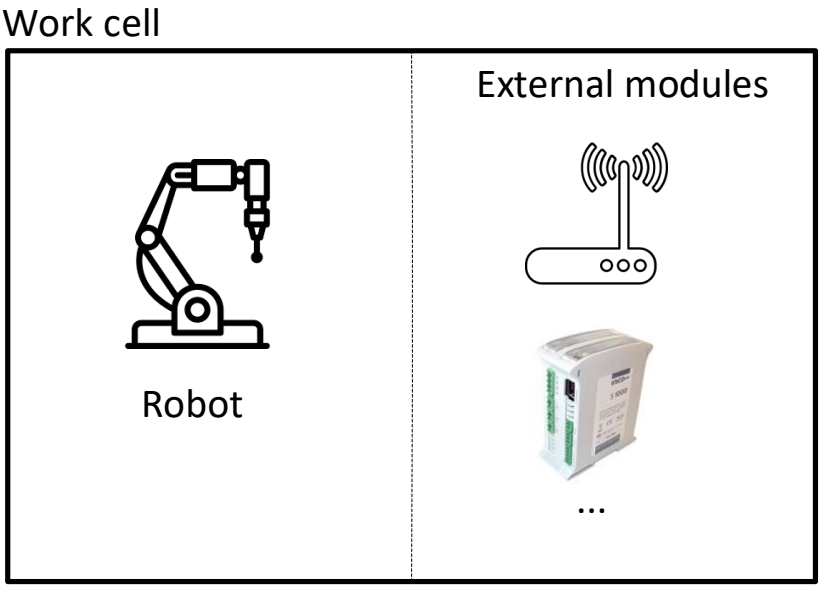


Figure 7. *Component diagram.*

Installation choices are affected by component capabilities and compatibility. Components should allow appropriate connections that correspond to needed real-time abilities and transferred data-structures.

3.1.1 Robot placement in cell

When considering robot position in work cell key issues are trying to avoid singularities and possible collision during movements. In some cases, singularities may be tried to avoid with mathematical calculation or adding small angles to the tooling. Angling tool may help in some cases but does not offer certain solution. Mathematical approach is more accurate, but can be very time consuming. This methodology exploits simulating software which many robot manufacturers offer as standard toolkit.

Simulating software approach is initiated with creating rough estimate of the actual work cell. Work cell, possible external modules in work cell, all robot work areas and needed robot tool locations are created into simulating environment. Then robot model is placed into work cell and all robot movements between work areas and other tool locations are simulated. If simulating software reports about joint singularities or collisions, robot location need to be changed. This process will be repeated as long as the correct robot position is found.

3.1.2 I/O wiring

When wiring inputs and outputs the key issues consider about choosing the right type of communication method between robot and modules. Choices made here affect straight to software development phase. Key factor for selecting communication method is the needed data complexity between communicating devices. Of course, the choices are limited to available communication methods. Some most used communicating methods and their preferred usage situations are introduced in table 1.

Table 1. *Industrial robots communicational methods and prefer usages.*

Communication method	**Preferred usage**
Digital I/O	Transferring two states, one sources.
Analog I/O	Transferring multiple states, one source.
Serial communication	Transferring simple binary data.
Ethernet	Transferring complicated data structures.

Each method mentioned in table 1 have different requirements. Each module that exploits digital and analog inputs/outputs need source power and ground wire from controller in addition of actual signal wires. Compatible voltage and current rates are crucial when connecting digital, analog and serial I/Os. Serial ports are in some cases realised with standard connectors, but main concern is assure that both communicational modules share the same communication protocol which can be separated in of two categories: parallel or serial. Each protocols have their own strengths and weaknesses. Parallel protocol is faster, but needs more wiring when serial protocol works with less wiring and is slower. Parallel serial connection created with nine wires is introduced in figure 8.

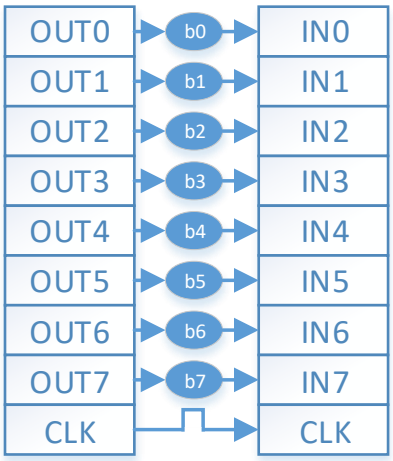


Figure 8. *An parallel 8-bit data bus.*

Serial interfaces send the data, one single bit at a time. The benefit of this transfer method is that for operational connection only two wires are needed. Serial interface in introduced in figure 9.

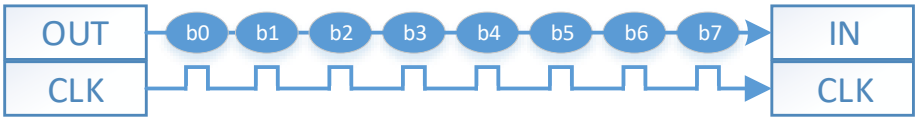


Figure 9. *Serial Interface.*

Ethernet communications usually exploits modular connectors when creating hardware connections. A Twisted pair cable with an 8 position 8 contact (8P8C) modular connector has become de facto standard in Ethernet connections. Connectivity to this transfer media is usually implemented to Ethernet connection supportive modules. If other connectors are used, the compatibility has to be ensured. Ethernet as a communication protocol benefits from massive, standardized protocol stack built on top of it.

3.2 API design methodology

When system hardware decisions are made it is time to start designing system on software level. Because the goal is to build service-oriented system, the process starts with identification of services that system need to provide. These services are later wrapped into API that works as interface for service users. The main criteria is to build system so it is able to offer services successfully in every situation. Key issues to consider are related to resource linking, minimizing complexity and clarity. Every module in system should serve well-defined purpose, so the API could be designed to serve that purpose.

This methodology exploits Unified Modelling Language (UML) through the design process. Use Case diagrams are used to identify needed API interface services. Component diagrams and class diagrams depict the relationships between software components and

sequence diagrams show how these components interact with each other in a particular use case scenario.

3.2.1 Identifying needed APIs

Things to consider when identifying API use cases are:

- What information and transaction would service user need?
- What information is available in module API is designed to?
- What services would service users need?
- Does user need to provide information so services could be accomplished?
- Is user-provided information needs to be saved? What module stores this information?

In these cases, the user refers to either end user or another module within the process. If system is going to have both end user API, and module-to-module APIs, design should be made from top down. First need to design end user API as clarified as possible and after the system supportive APIs. As a product of identifying process, a use case diagram is created.

3.2.2 API designing

After creating use case diagram the process continues with designing how those APIs are going to be implemented. When real-time demands, needed information flow and executed tasks are identified, the system need to be designed so it could respond to those needs. Through the design process a high abstraction level need to be applied so result models are going to be non-platform dependable.

UML offers tools for modelling the system and sub systems in high level of abstraction. These models may do this by hiding or masking details, visualizing components and connections between them. This methodology models system structure using component diagrams and object diagrams. These diagrams offer a way to present the physical aspects of object-oriented system. They can be also used to visualize, specify and document component based systems. Component diagrams focus on a system's components that are usually used to model the systems static implementation view. Example of a component diagram is presented in figure 10.

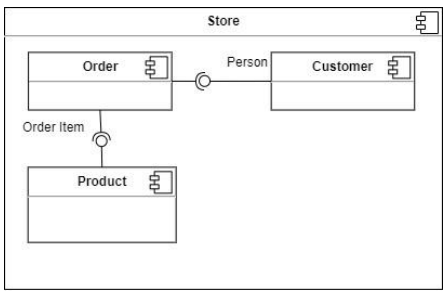


Figure 10. *Example of a component diagram.*

In the Unified Modeling Language a class diagram, like component diagram, is categorized as a static structure diagram. Class diagram presents the structure of a system by

showing the system classes, class attributes, operations and the communication among objects. Class diagram may present inheritance, associations and dependencies between classes. In additions class diagram offers information about class multiplicity, operation visibility and class attributes. Class diagram offer frames for later created interaction diagrams. Example of a class diagram is introduced in figure 11.

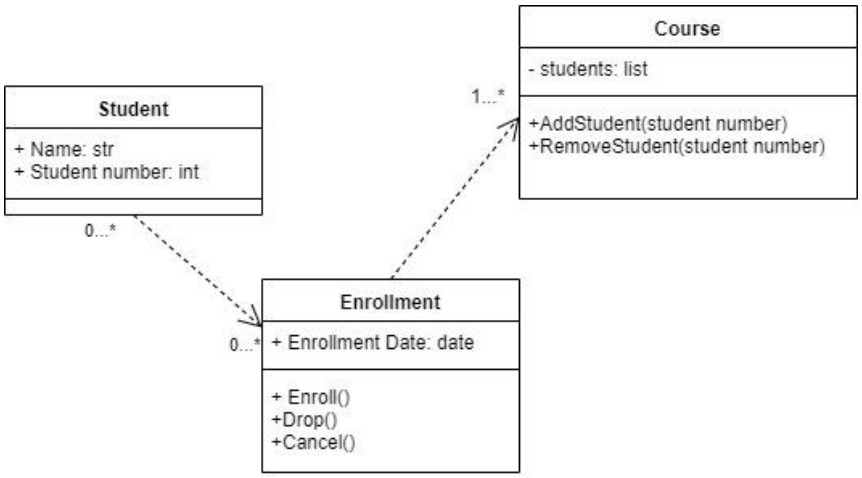


Figure 11. *Example of a class diagram.*

After objects are identified the sequence diagrams will be designed to model use cases scenarios. Sequence diagrams are interactions diagrams that present the execution order of operations in a use case. They capture the interactions between objects in the context of a collaboration. These diagrams form needed time line for functionality execution. An example of sequence diagram based on class diagram in figure 11 is introduced in figure 12.

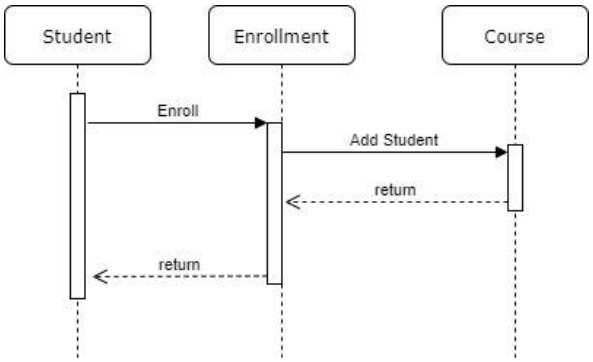


Figure 12. *Example of sequence diagram.*

4. IMPLEMENTATION

In this chapter presents the solution to accomplish the final system and solve the problems mentioned in chapter one using methodology introduced in chapter three. This includes all the designing steps in chronological order they were carried out during the deployment process. Chapter starts with introduction of the current manufacturing line setup and the reused components. Next is presented the hardware decisions like robot placement in cell, tool attachment to robot flange and wiring of sensors and other I/O. Third the system architecture in communication and use case level is introduced. After architecture design the implementation of these use cases in software level is presented. Finally chapter exhibits the python script that converts SVG-images to point-lists that Robot can read for drawing process.

Implementation phase is carried out using the methodology presented in chapter three. Chapter 4.2 matches the methodology in chapter 3.1 and chapter 4.3 matches to methodology in chapter 3.2.

4.1 Manufacturing line

The improvements are done to the current manufacturing line that consists of twelve work cells interconnected with conveyor belts. Excluding cell seven and one, all the cells include the robot, SONY SRX-611 and penholders for three different pens. These work cells are designed to perform the same functionality. They are used to draw different mobile phone components to paper carried by pallets on conveyor belt. The cell one is designed to supply the system pallets with new papers and removing the final products from the pallets. New pallets are fed and removed to production line by the user in work station seven. The simulation model from the manufacturing line is presented in figure 13.

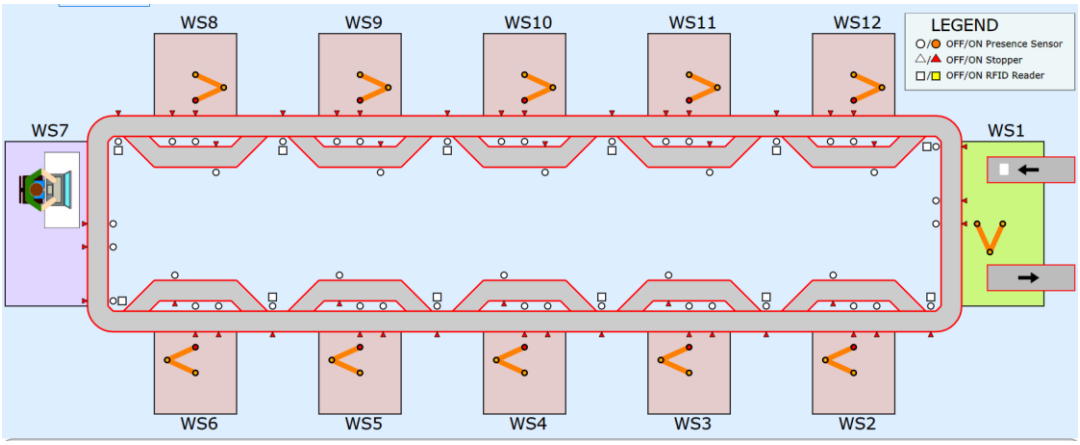


Figure 13. *Manufacturing line simulation mode.*

The main purpose of the manufacturing line is to simulate the mobile phone assembly line. Instead of using real components, work cells draw the components to paper. Each robot is designed to draw three different variations of three different mobile phone components (screen, frame and keypad). Each component can be drawn with three different colours. This is achieved with special pneumatic controlled tool that is able to grip on a pencil and control the vertical pressure that is directed to pencil tip while drawing.

Each work cell is equipped with acrylic safety doors to prevent operator being in contact with working robot. Safety system includes interlock door switches and emergency buttons that are connected to safety relays.

The basic work cycle starts with cell seven loading paper to pallet. After paper loading the pallet starts moving through the main line. Pallet could be guided either into the work cell where it is stopped with pneumatic stopper for drawing, or it could bypass the work cell with conveyor belt going around the work cell. Pallet continues to do so with every robot work cell until it reaches again cell seven where the paper is picked up and stored.

Manufacturing system is controlled with web service architecture based interface. Each work cell and conveyor belt works as a service provider. By waking these services the Manufacturing execution system is able to control the system in various ways. System architecture allows the hardware and software changes without the need to reconfigure MES (Manufacturing Execution System) program. All possible robot-controlling services are presented in table 2.

Table 2. *Web services provided by each work cell.*

Service	**Service description**
Draw*	Robot draws a preload picture *1-9 to pallet.
GetPenColour	Returns the colour of current pen
ChangePenRed	Robot changes the pen color to red
ChangePenGreen	Robot changes the pen color to green
ChengePenBlue	Robot changes the pen color to blue

Web service interface is created with Inico s1000 remote terminal units. Each work cell and each conveyor belt section has RTUs that controls the manufacturing system section when the web services are waked with HTTP requests. RTUs communicate with robots with serial communication. System communication architecture is presented in figure 14.

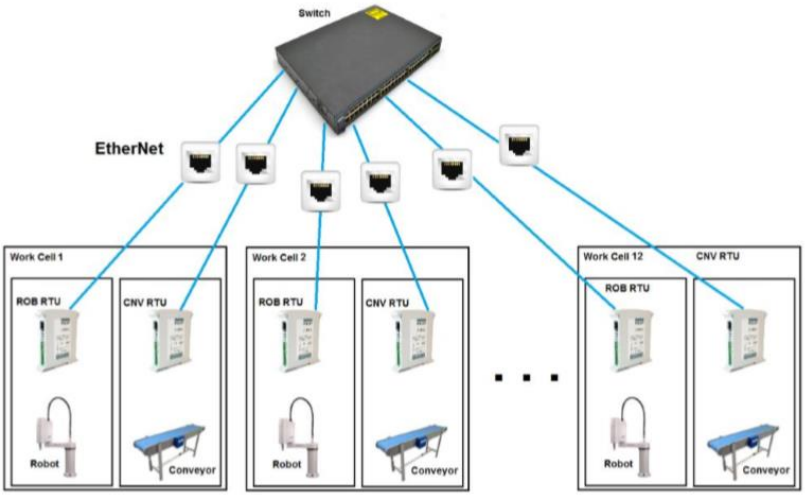


Figure 14. *Communication architecture of the system.*

The current manufacturing line in being improved by replacing the old SCARA-type robots with new 6-axis factory robots. These new robots include advanced communication protocols, built-in pneumatic valves and possibly more degrees of freedom that can be used in applications to come. Simultaneously with the robot replacement the safety system is being replaced on the manufacturing line.

4.2 Hardware design

4.2.1 Omron Viper 650

Installed robot is six-axis robot model viper 650 manufactured by Omron. It is designed for machining, assembly and material handling. Viper 650 has absolute encoders with high resolution to provide easy calibration and high accuracy. The maximum reach of this model is 653mm and it is able to carry payloads up to 5kg.[47] Robot is controlled by eMotion Blox-60R controller. Robot is presented in figure 15.

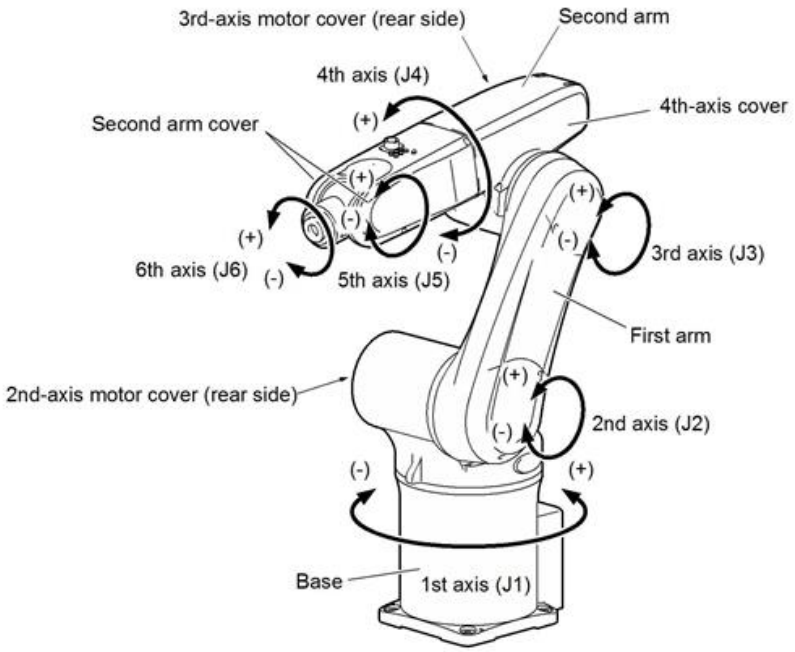


Figure 15. *Omron Viper 650 assembly robot. [47]*

Robot comes with controller, front panel, T20 pendant required connection cables and a CD that includes the Automation Control Environment (ACE), and simulation environment made by Omron. These elements form the robot control system. The high level connections between these units are presented in figure 16.

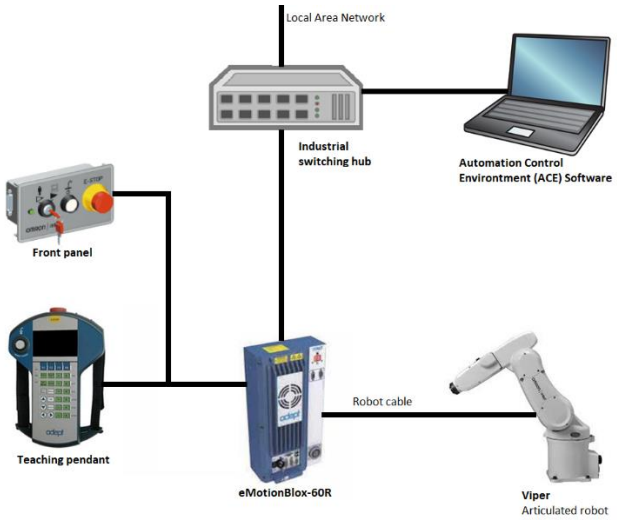


Figure 16. *Robot control connections.*

Viper 650 system includes Teaching pendant that features emergency stop switch and three-position safety switch that prevents pendant input or robot motion when switch is not engaged. Software features include location teaching, robot jogging, digital I/O

control, robot status display and error messages.[47] So pendant can be used for teaching locations but for more complex task programming ACE is mandatory.

The programing environment ACE provides an environment to program, emulate and deploy applications. With emulation environment and 3D visualization, it is possible to run quick proof of concepts without any actual robot hardware. ACE runs on PC and is connected to robot controller with IP protocol. ACEs program editor provides online programming tools for eV+ and C# programs. With Task Status control user is able to monitor and control current tasks and exceptions. Most of these features are in use only when application is executed from ACE software. It is also possible to upload application to robot controller so external controlling PC is not mandatory. The Graphical User Interface of ACE is presented in figure 17.

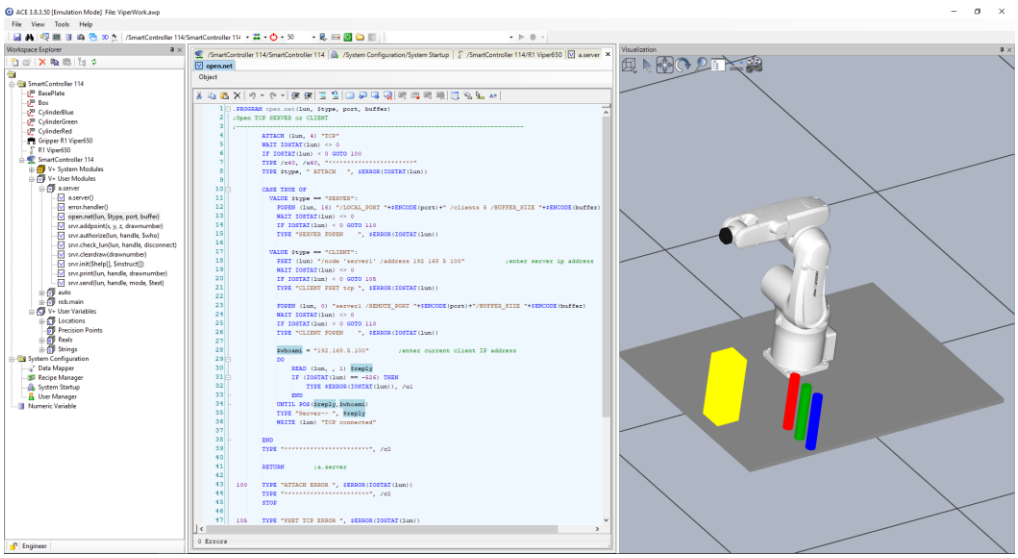


Figure 17. *Graphical User Interface of ACE programming environment.*

4.2.2 Robot placement

First methodology goal was to decide the correct position for robot in the cell. The goal was to use a position from where the robot could easily reach to needed locations and move between them without the danger of joint singularity or collisions. The software used to decide robots location in work cell was ACEs emulation environment. The used emulation environment is presented in figure 18.

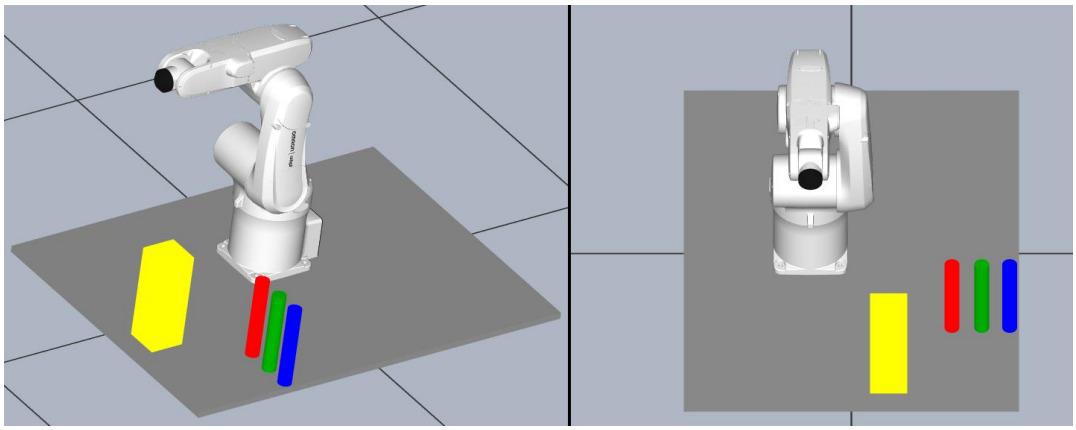


Figure 18. *Simplified emulation environment of the robot cell*

For creating responding 3D environment the measurements was taken from the Cell floor, the positions of the drawing plate and pen holders. In figure 18 the yellow box represents the drawing surface and cylinders represent the pen holder positions. When 3D environment was ready the robot was jogged in emulation mode to every possible location and between every possible location transform to decide the robot position in the working cell. Another critical factor was the need to keep the orientation of the 4th axis constant due to the connectors on top of second arm. This prevents 5th axis changing its orientation to mirrored side which could later cause problems with sensor wiring. During the simulation, it turned out that the robot was able to perform correctly as long as the base was mounted far enough to the left side of the work cell floor. This allowed the 4th axis to keep its orientation almost constant through all the movements between needed locations. The final position was decided to most centre position as possible to keep the pallet and pens as close as possible from the robot. In this robot position, no singularities occurred during any work movement of the robot. The final orientation of robot is presented in figure 19.

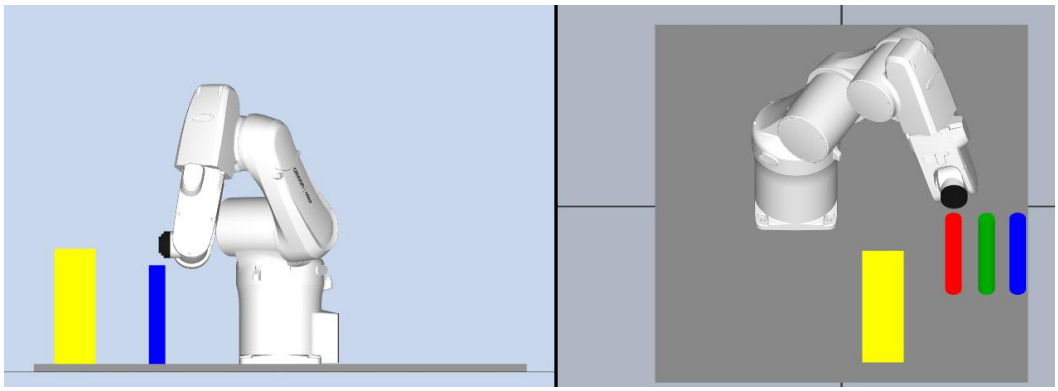


Figure 19. *Final robot joint orientation.*

Robot was mounted to work cell floor with special designed baseplate. Baseplate consisted of two steel plates welded together with steel shafts. Bottom plates of steel had predrilled holes that matched the pre drilled holes in the work cell floor. The top plate had holes which was used to mount the robot to the top plate. Baseplate had different possibilities to mounting the robot to it. This made possible to move robot on the plate later if needed. Plate also raised the robot 15cm from work cell floor to compensate the tool location difference

4.2.3 Tool attachment

Due to different robot tool attachment method compared to old robot, there was need to design a part to allow the tool to be attached to robots flange. This step is not included in the methodology in chapter three, but it is documented for future development purposes. Goal was to reuse the existing gripper that has been worked well in previous application. Tool was steel machined gripper that has two pneumatic cylinders. One is used to move gripper vertically and other is used to open and close the gripper claws. Tool has three different sensors. Two to sense if the cylinders are open or closed and one to sense the present of the pen in gripper. The tool is presented in figure 20.

Figure 20. *Robot tool.*

The design of the attachment part was created using CAD software Solidworks. The measures was taken from tool attachment shaft and flange bolt spacing were checked from robot manual. Designed part needed to attach the tool to 90 degrees of the flange surface so the axis orientation mentioned in chapter 4.2 could be accomplished. Drawing of final attachment part is presented in figure 21.

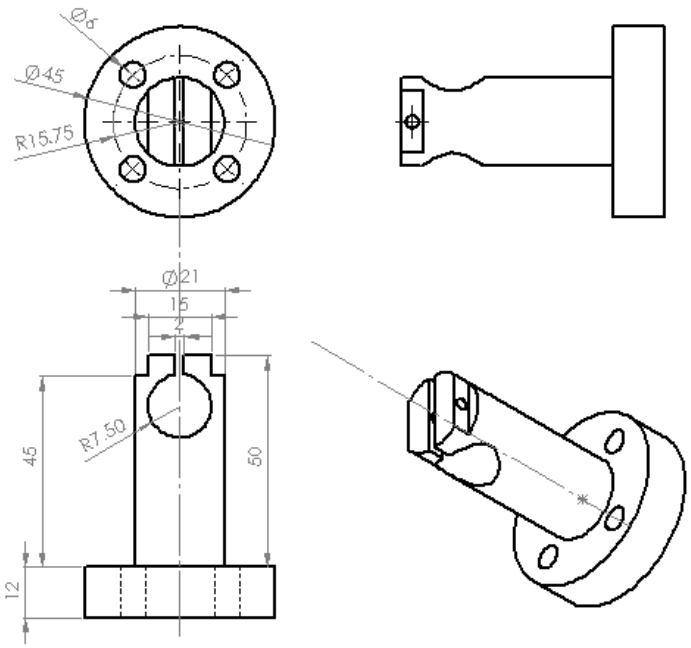


Figure 21. *Drawing of tool attachment part.*

After the design was completed, the part was 3D-printed with PLA plastic to create the prototype of the attachment part. The PLA plastic provides the stiffness and flexibility to hold tool tight in place. In other hand, it works as safety wrist at the collision situations preventing other more expensive parts from damaging. Final 3d-printed part is presented in figure 22.

Figure 22. *Tool attachment part.*

Next phase of methodology presented in chapter three would be decisions considering about the supportive modules. Since the nature of this system improvement is to maintain existing supportive hardware, this phase is skipped during design process. As mentioned in chapter 4.1 the Inico s1000 is used as supportive module to create RESTfull interface for system users. Because s1000 and Viper 650 both support Ethernet connection, the wiring design is not needed between these modules. Since module decisions are already made, next phase is robot-wiring design.

4.2.4 I/O wiring

After robot was mounted to desired location it was time to connect tool sensors and pneumatic cylinders to robot so they could be controlled with control program. This meant the need to create needed connections from controller to robot interface panel and from robot second arm connectors to tool. The position of second arm connectors is marked in figure 23 with letter “A” and the location of robot interface panel is marked with the “B”.

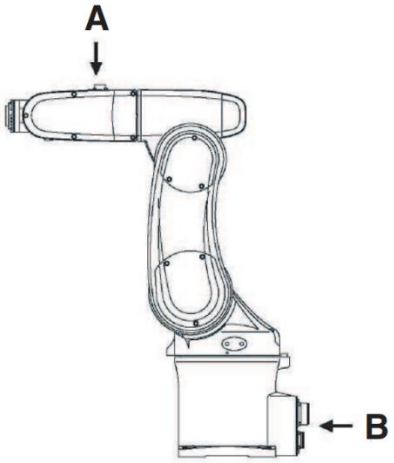


Figure 23. *Locations of robot interface panel and second arm connector[47]*

Robot interface panel includes two pneumatic connectors, two signal connectors and grounding terminal. First pneumatic connector(AIR1) offers pneumatic pressure for three solenoids in robot. These solenoids can be controlled with control program. The second pneumatic connector(AIR2) connects directly to AIR2 on the second arm. Connector named CN22 is used to provide power and signal between the eMB-60R controller and the robot with premade cable. Connector CN20 is 20pin round type connector which pins one to ten are connected to matching pins from one to ten on CN21 connector on the second arm. Pins 12 to 19 are used to control the solenoids.[47] The locations of connectors on the robot interface panel are presented in figure 24.

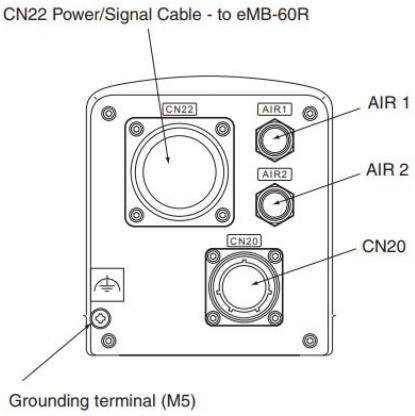


Figure 24. *Description of connectors on robot interface panel[47].*

On top of the second arm is located seven pneumatic connectors and one ten pin round type signal connector connected directly to connector CN20 pins one to ten. Six pneumatic connectors are workings as pairs. Depending the solenoid controls, one of the pair is working as air intake and one as air exhaust side. Seventh pneumatic controller called AIR2 is connected straight to AIR2 connector on the robot interface panel. Locations of second arm connectors are presented in figure 25.

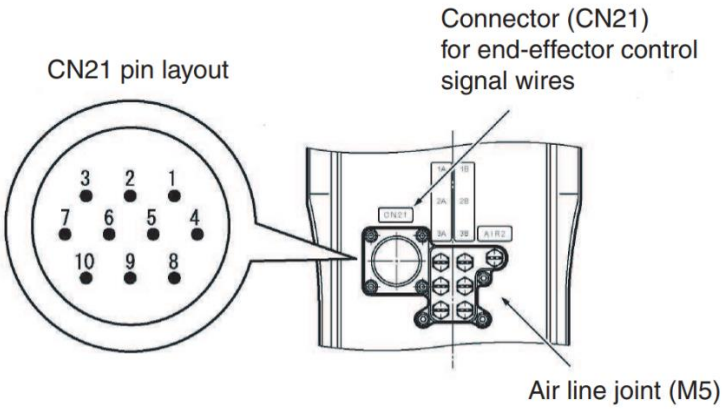


Figure 25. *Second arm connectors[47].*

Third important connector base is located on the eMotionBlox-60R controller. Controllers interface panel includes connectors for Ethernet, smart servos, 24VDC input, VAC input and variety of I/Os for prefixed and user use. All connectors on controller interface are presented in figure 26.

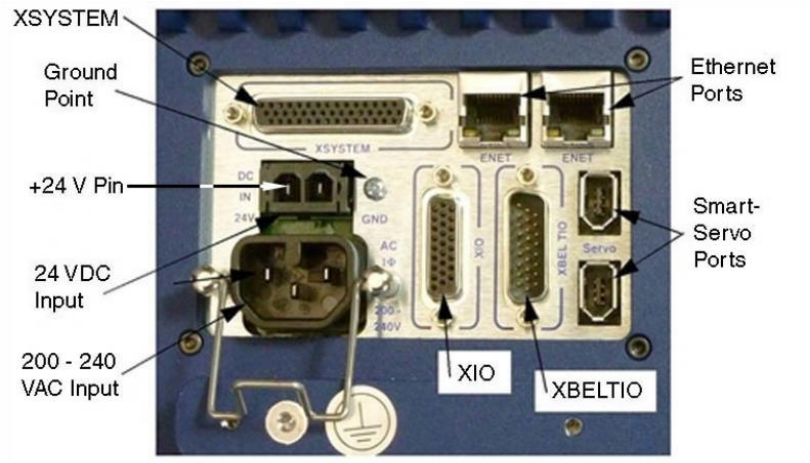


Figure 26. *Connectors on eMotionBlox-60R interface panel.*

Controller interface panel connector XIO offers I/O signals for external devices. XIO connector provides 12 inputs and 8 outputs. The eV+ program is able to use these signals to perform functions in the work cell. The 12 input channels are divided in two groups of six. Both groups are electrically isolated from the other group and is optically isolated from the eMotionBlox-60R ground. All inputs on each group have a common source/sink line. These inputs can be accessed with direct connection to the XIO connector. The specification of the XIO connector is presented in table 3.

Table 3. *XIO connector specification [47].*

Parameter	Value
Operational voltage range	0 to 30 VDC
"Off" state voltage range	0 to 3 VDC
"On" state voltage range	10 to 30 VDC
Typical threshold voltage	V_{in} = 8 VDC
Operational current range	0 to 7.5 mA
"Off" state current range	0 to 0.5 mA
"On" state current range	2.5 to 6 mA
Typical threshold current	2.0 mA
Impedance (V_{in}/I_{in})	3.9 KΩ minimum
Current at V_{in} = +24 VDC	$I_{in} \leq$ 6 mA
Turn on response time (hardware) Software scan rate/response time	5 µsec maximum 16 ms scan cycle/ 32 ms max response time
Turn off response time (hardware) Software scan rate/response time	5 µsec maximum 16 ms scan cycle/ 32 ms max response time

Before making any connections there was a need to make sure that voltage and current ranges were corresponding with XIO connectors range. Sensors used in tool are D-M9P and D-A93 auto-switches produced by SMC Corporation. Sensor used to detect pen on tool is UM-R5TVP miniature sensor produced by Takex Corporation. All of the sensors operate with 12-24 VCD Voltage and 5-50mA load current range so they are compatible with robot I/O system.

When the compatibility of sensors and controller were verified, it was time to design connection cable. First cable was designed between XIO and CN20 connectors. Voltage, ground and three input channels were brought to CN20 connector pins between one to eight. This offered voltage to input channels for sensors, since first ten pins of CN20 connector were connected straight to connector CN21 corresponding pins. With this approach the sensor signals were available from software using eV+ signal numbers 1097-1099. XIO outputs one to six were connected to CN20 pins 13-18. These offer control signal for solenoid controller pneumatic valves using eV+ signal numbers 97-102. Table of all robot connections is presented in appendix A.

4.3 Software design

High level system architecture of the manufacturing system under modification can be defined using special features like distributed control architecture, open field communication, and the hard real-time requirements. These requirements need to be met in order to reach a successfully developed application. Just described features require certain needs in the process of application development. Partitioning, interoperability and allocation of the application parts to the system resources have a significant role in the process of application development.

Device interoperability refers to the need of common networking media, protocols and data formats between interconnected devices. In the manufacturing system under modification, these challenges are solved with using TCP/IP connection between all interconnected devices and by creating uniform API's between different levels of application. This design makes system modular, scalable, flexible and easier to test. High-level system architecture is presented in figure 27.

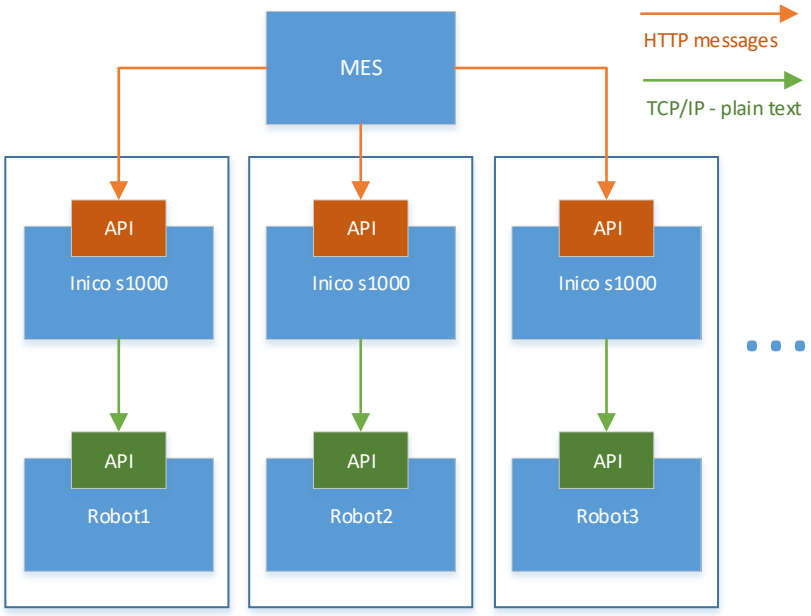


Figure 27. *High-level software architecture diagram.*

Due to systems strong service orientation and distributed architecture, the component based and service-oriented approach is used in development. The process start with creating set of design-level and implementation level use case-, class-, and object diagrams described in chapter 3.2. Behavioural diagrams like sequence diagrams are used to define methods of those classes at different levels of detail. All design models are created platform independently based on feature knowledge of the devices involved. From figure 27 we can identify three main components in system. Manufacturing execution system, Inico S1000 RTU and factory robot. Next chapter presents Inico S1000 specification and functionalities. It also introduces created models to specify RTUs functions. These models are made using UML (Unified Modelling Language).

4.3.1 Inico s1000 software design

Inico s1000 is a programmable Remote Terminal Unit (smart RTU) device, which offers process control capabilities, as well as a Web-based Human-Machine Interface (HMI) and support for Web Services. This RTU is equipped with multiple connectivity methods, web-based Structured Text editor with syntax highlight and built in compiler. All the internal programs are programmed using IEC61131-3 Structured Text programming language [48]. Inico s1000 device is presented in figure 28.

Figure 28. *Inico s1000 RTU.*

In this application, the RTU is used for creating a RESTfull interface for MES application usage through Local Area Network. Inico s1000 is capable of launching designated functions as result of certain HTTP-requests. When user sends request to RTU, RTU communicates with robot to execute service and messages back to user information about completing the service. This is fulfilled using Inico s1000s internal event triggering ability. Since Inico s1000 is not object oriented environment, the software methodology section will be executed only partial. To accomplish successfully functionality, only use-case and component models are needed.

The services offered to MES applications are listed in chapter two. A use case diagram based on needed services is presented in figure 29. Diagram shows the functionality Inico S1000s needs to introduce to maintain successful application.

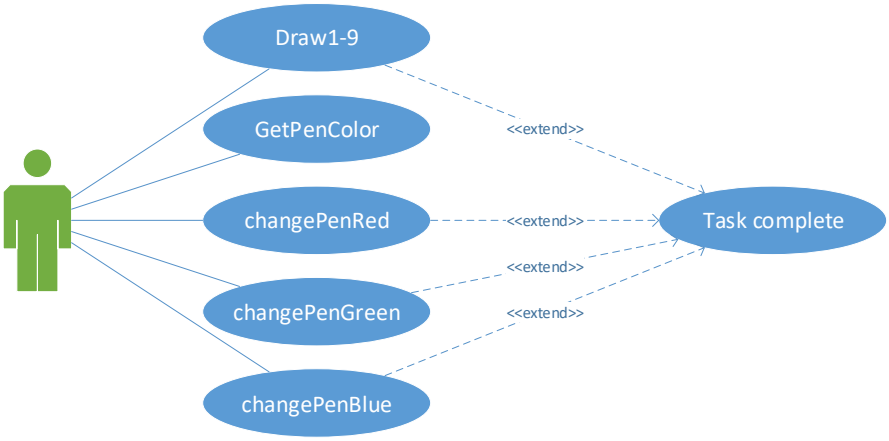


Figure 29. *Use Case diagram for Inico s1000.*

Based on this Use Case diagram component specification model of Inico S1000 was created. Due to RTUs only purpose of introducing interface, commanding robot to start tasks and informing user when tasks are finished, RTU does not hold stationary information. Its only purpose is to translate robots messages to different form through RESTfull API. Component specification model of Inico S1000 is presented in figure 30.

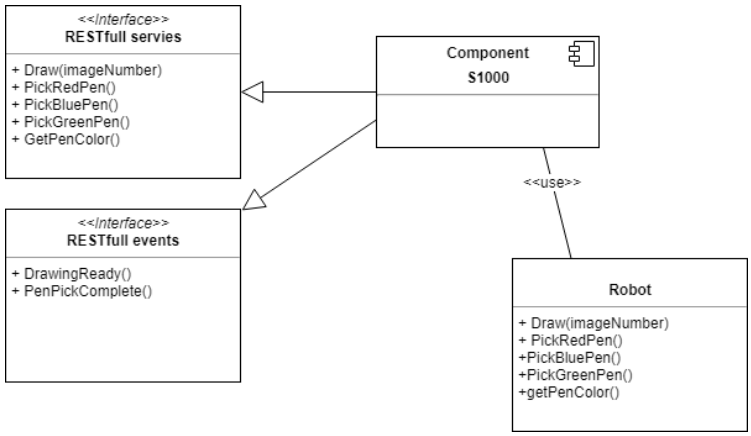


Figure 30. *Component specification model of Inico S1000.*

4.3.2 Robot software design

Use Case diagram presented in figure 29 can be also used when designing robots component model. Due to the Inico S1000 acting as a message broker, same use cases apply also for robot. In addition to these use cases there needs to be a way to upload drawings to robot memory and configure drawing frame used to draw pictures. Robots Use case diagram is presented in figure 31.

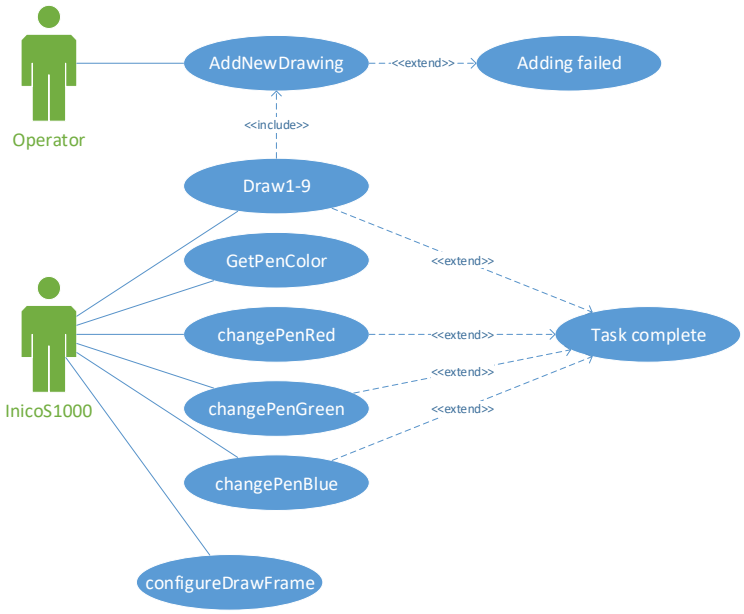


Figure 31. *Robot Use Case diagram.*

As robot is doing the actual work drawing and picking pens we need to find out the subcomponents taking part in this process. Robot needs to have offline programmed positions, data store for managing drawing points, a tool that offers sensor data and error handler that stops processes if Emergency button is pressed. The component specification model is presented in figure 32.

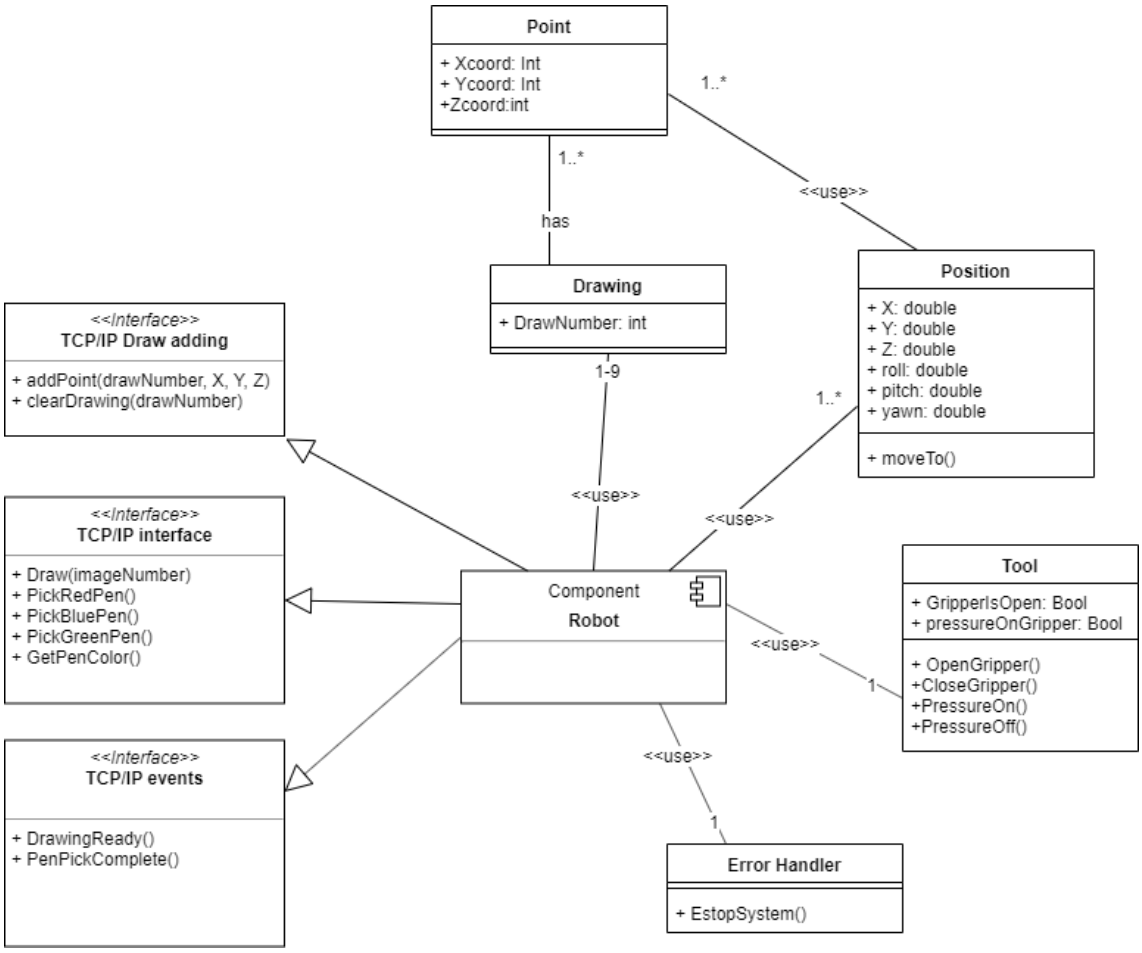


Figure 32. *Robots component specification model.*

After declaring the basic subcomponents for robot the design models to represent the functional flow during the use cases were created. Process started with creating simplest information transfer case where pen colour is transformed from robot to MES-level. Sequence diagram for getting pen colour is presented in figure 33.

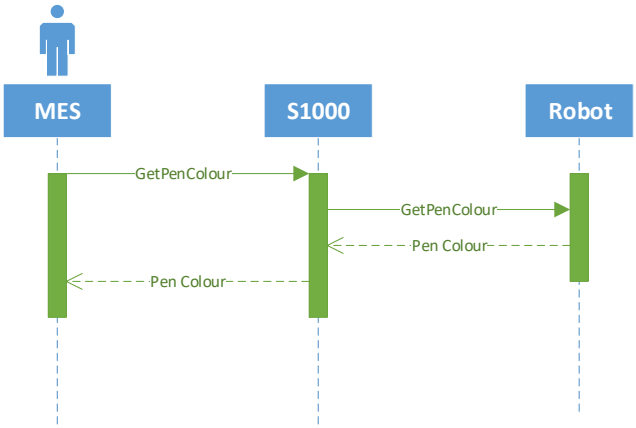


Figure 33. *Sequence diagram for getting pen colour.*

Next sequence diagrams are presented in robot subsystem. In drawing task robot needs to switch off the pressure from tools vertical cylinder so the tooltip is able to move freely vertically during the drawing process. This approach lengthens the pens lifespan and ease keeping the pen tip on paper during the drawing process. In drawing process robot moves through list of predefined points of drawing. The sequence diagram of drawing process is presented in figure 34. With this diagram, we are going to define needed sub functions and execution order for drawing process in implementation phase.

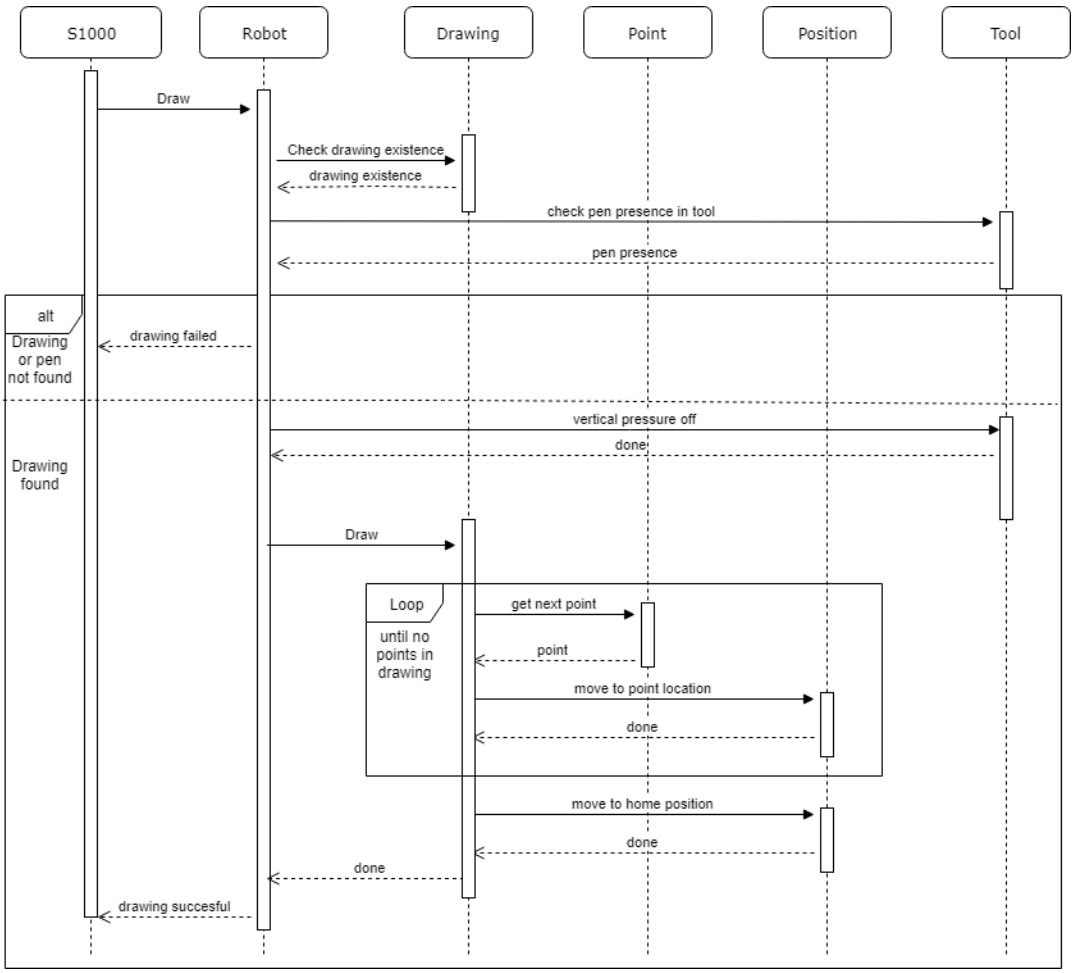


Figure 34. *Sequence diagram of drawing process.*

Use cases where robot changes the pen on tool are going to have similar execution programs. The difference between these programs are going to be the positions the pens are going to be picked. During these processes the tools vertical movement needs to be minimized to provide repeatable positioning during the process. Robot needs to drop off the existing pen on tool before getting a desired one. The sequence diagram of pen picking process is presented in figure 35. If user tries to change pen colour to existing one, there are no need for movements of the robot.

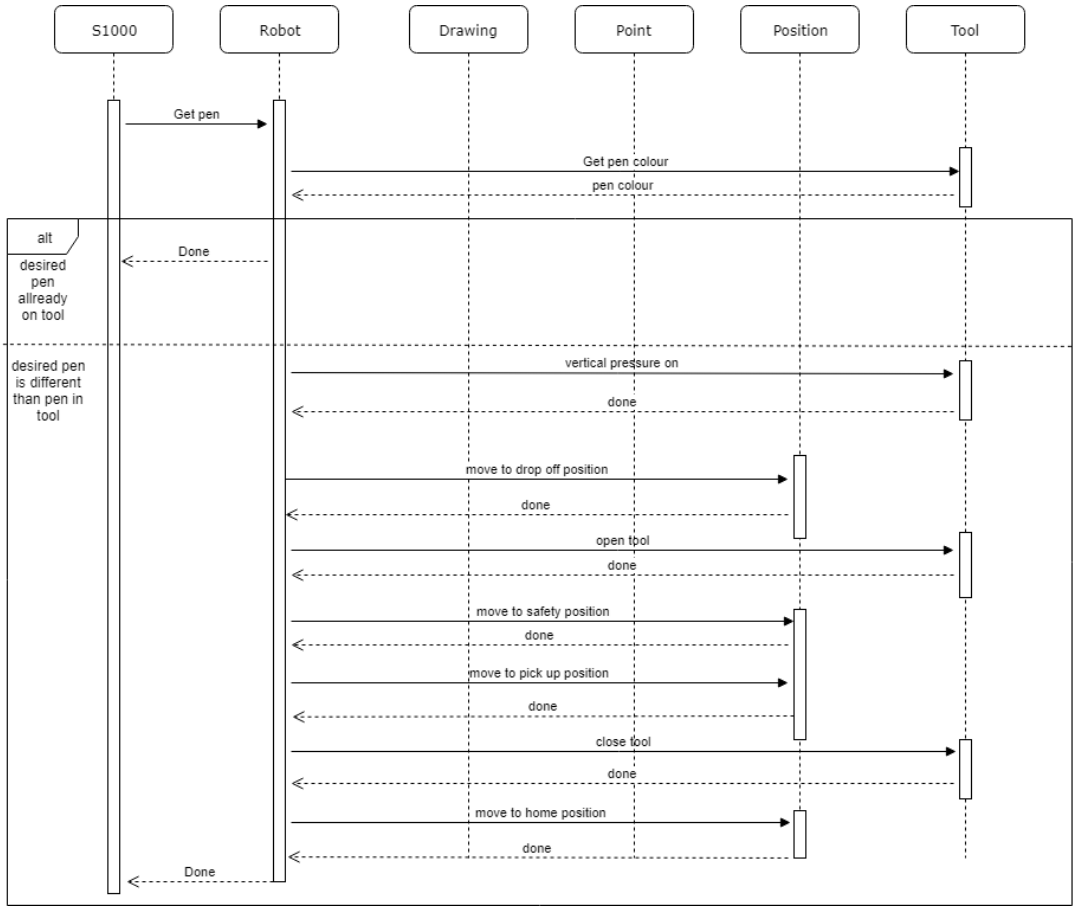


Figure 35. *Sequence diagram of pen picking process.*

Third essential process is levelling the frame used in drawing process. Drawing frame is placed to pallets surface x-axis facing crosswise and y-axis lengthwise the pallets surface. The orientation can be rearranged by relocating three of the calibration points in the system.

The purpose of draw frame calibration process is to ensure that the frame surface is in line with the pallets surface. In configuration process, the presence sensor on robots tool is used. This sensor offers information about whether the vertical pneumatic cylinder is on outside position or not. By using this sensor, the robot is able to detect when the tool tip is touching the surface while cylinder is not pressurized. The sequence diagram of frame configuration process is presented in figure 36.

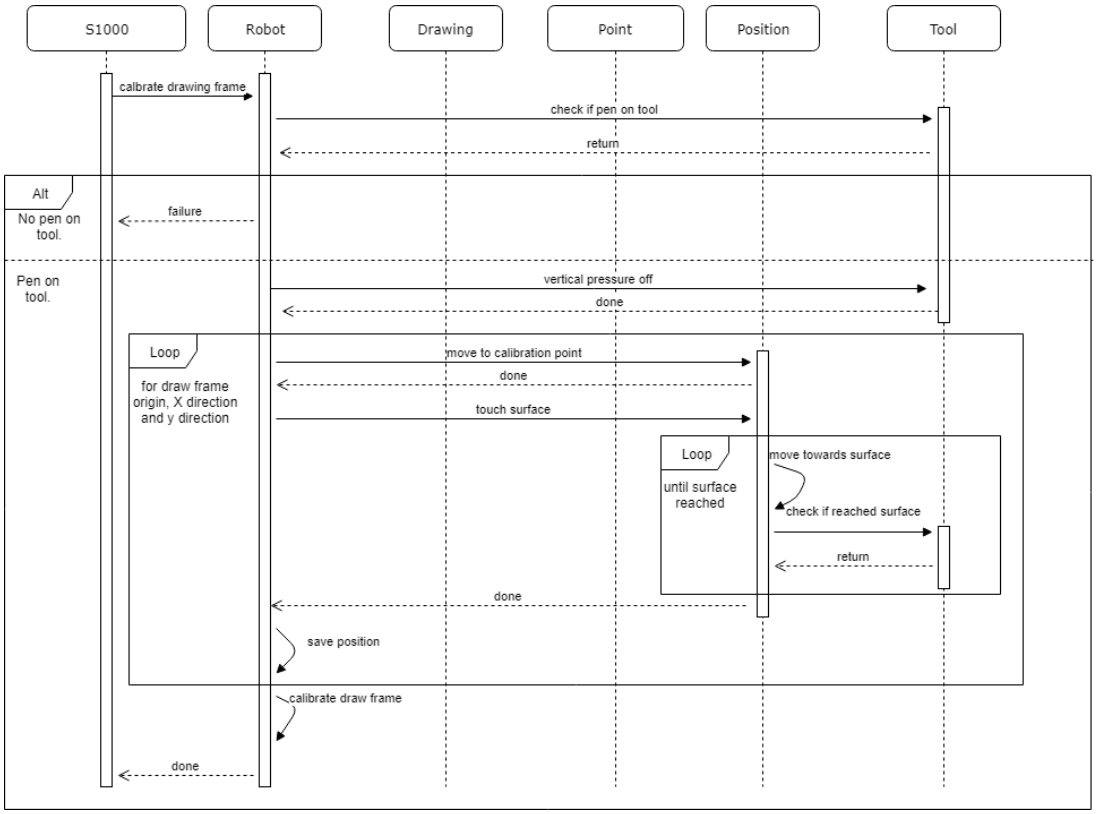


Figure 36. *Sequence diagram of drawing frame calibration process.*

4.3.3 Image transfer sequence

Fourth essential process is drawing addition process. Since robot does not support any data-interchange formats like JSON or XML there was need to find a way to exchange drawing point data from user to robot using TCP plain text. To keep interface simple enough the decision was made to transfer drawing data point by point. Though this might be time-consuming approach the process is done so unfrequently it should not affect the user experience. The sequence diagram of draw addition is presented in figure 37.

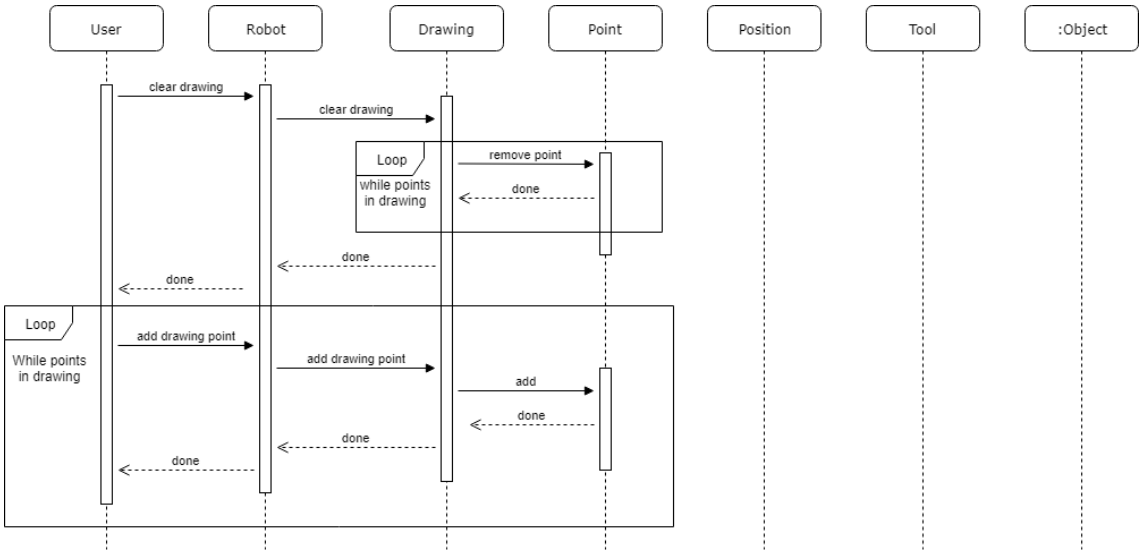


Figure 37. *Sequence diagram of draw transfer process.*

4.4 Implementation of design

On this phase the system design is transferred to actual programs in devices. This includes programming robot's functional programs, robot's TCP-server and Inico S1000 RTU programs that communicate with robot's API.

4.4.1 Inico S1000

Inico S1000 communicates with robot with TCP plain text messages. Certain outgoing messages start robot functionality and certain incoming messages offer information if task was successfully executed. All TCP messages and their purpose are listed on table 4. RTU uses one digital signal input to detect if robot is currently busy. If this event occurs, RTU does not send TCP message but inform user about the situation thru RESTfull API.

Table 4. *Inico S1000 TCP plain text messages.*

Message type	**message**	**message purpose**
Outgoing	pickPen1#	Change to pen 1.
Outgoing	pickPen2#	Change to pen 2.
Outgoing	pickPen3#	Change to pen 3.
Outgoing	draw1*	Start drawing *number 1-9 specifies the image to draw.
Outgoing	configure	Configure draw frame.
Incoming	pickPenStarted	Operation of change pen has started.
Incoming	pickPenEnded	Operation of change pen has ended.
Incoming	drawStarted	Operation of draw has started.
Incoming	drawEnded	Operation of draw has ended.

Inico S1000 designed system architecture consist of structured text programs that correspond with services provided by RESTfull API. When a service is waken with HTTP-request runs RTU the ST-program designed to send command TCP-message to robot. In addition to service waking programs one looping ST program listen incoming TCP-messages and triggers events informing MES-level about task progress. The data transfer architecture is presented in figure 38.

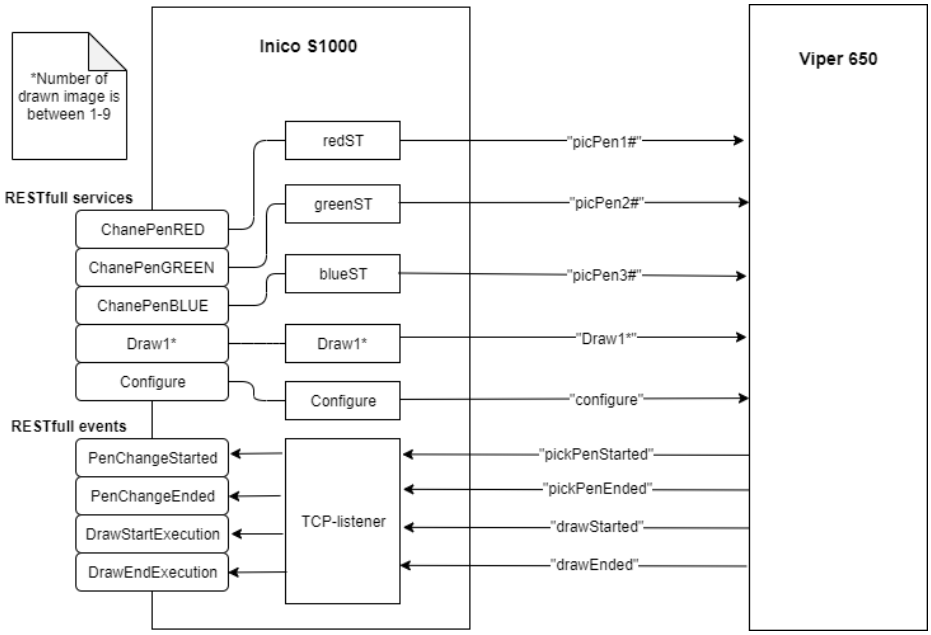


Figure 38. *Message transfer between RTU and robot.*

4.4.2 Viper 650 robot

The basic execution system of Omrons Ev+ programming environment consist of created programs and tasks where those programs can be executed. User has in total of 24 different tasks to execute programs in simultaneously. In designed system Viper 650 factory robot has two main roles. To work as a TCP-server for RTU client and execute the physical tasks commanded thru TCP-connection. Due to the need of continual execution of the TCP-server the server program is going to run on its own task. This way TCP-server program is able to listen messages and launch robot controlling program on another task. This TCP-server program forms the actual APIs for RTU and drawing addition.

All in all 26 different programs where created to implement robots software design. The programs can be separated in two groups. Functional and informational group. Functional group includes all programs that affect robots physical presence including movements, actuator controlling and sensor reading. The physical programs are enclosed under “rob.main” program package. Informational programs include TCP-server

and all programs that support its actions. TCP-server, draw clearing and draw point adding are enclosure under “a.server” program package. Third independent program error.handler() monitors robots E-stop channels. In case of E-stop presence, error handler terminates all functional programs and allow continuation when E-stop is released. Robots informational functions are presented in table 5.

Table 5. *List of robots informational programs*

Function name	**Function objective**
a.server()	Polls TCP-connection and launches functional programs.
srvr.addpoint(x,y,z,drawnumber)	adds new drawing point to drawing
srvr.cleardraw(drawnumber)	clears the draw
srvr.init()	Initialize TCP server
srvr.authorize(lun,handle,who)	checks if client IP adress is authorized to connect
open.net(lun,type,port,buffer)	Opens TCP connection
srvr.print(lun, hadle, drawnumber)	prints all point coordinates of a drawing
srvr.send(lun,handle,mode,text)	sends message to client
srvr.check_lun(lun,handle,disconnect)	Checks logical unit condition.

To accomplish needed movements user needs to program needed locations offline. These locations are robots home location, three different locations for pen pickup, the frame used to draw images on pallet and safety location used between pen drop off and pick up. Using these locations robot is able to accomplish all needed movements.

We are going to build robot functionality from bottom up. Programs are created using sequence diagrams from previous chapter. All functional programs and their objectives are presented in table 6. Some functions are used to reach the desired positions, some control robot outputs to control tool and some change robots internal variables.

Table 6. *List of robots functional programs.*

Function name	**Function objective.**
rob.close_tool()	Close gripper.
rob.draw(runpoints[,])	Move through all attribute lists drawing points.
rob.draw_conf()	Run drawing frame configuration.
rob.drop_pen()	Select correct function to place existing pen back.
rob.drop_pen1()	Drop pen off to location one.
rob.drop_pen2()	Drop pen off to location two.
rob.drop_pen3()	Drop pen off to location three.
rob.flow_off()	Set vertical pressure of tool off.
rob.flow_on()	Set vertical pressure of tool on.

rob.go_home()	Move to home position.
rob.init()	Initialize robots attributes at startup.
rob.open_tool()	Open gripper.
rob.pick_pen1()	pick pen from position one.
rob.pick_pen2()	pick pen from position two.
rob.pick_pen3()	pick pen from position three.

Functions to open and close tool gripper change the output signals 99 and 100. When pneumatic valve controlled by signal 99 is closed and valve controller by signal 100 is open,

e forces pneumatic cylinder to open the gripper. When signals are in opposite phase, gripper closes. Same method is used with tools vertical movement with the difference that only one, the cylinder thrust side is connected to pneumatic system. This way vertical cylinder can have either stiffing pressurized state or unpressurized flowing state.

In programs created to pick and drop off pen we use sensor signals to makes sure that process is progressing correctly. Signal 1097 is true when gripper is holding a pen. Signal 1098 is true when gripper is open. Using these signals we make sure that robot tool is functioning correctly and we do not pursue movements that may cause collisions. Example of signal usage is presented in program 1. First on row 31 we check if gripper is holding a pen. If needed the pen is placed to holder. After tool opening command on row 37 we need to make sure if gripper has actually opened. If pneumatic system is malfunctioning and gripper is still in closed state. The information is sent to pendant screen in row 41. Gripper malfunctioning prevent the pick process to start in row 45. Similar method are used every time gripper is operated with pneumatics.

```
30              ;if pen present on gripper
                IF SIG(1097) THEN
32                  CALL rob.drop_pen()
                END
34
                ;make sure that there is no pen on gripper
36              IF NOT SIG(1097) THEN
                    CALL rob.open_tool()
38                  WAIT.EVENT , 1

40                  IF NOT SIG(1098) THEN
                        PDNT.NOTIFY "Pneumatic problem", "can't open gripper"
42                  END
                    ;start pick process
44                  IF SIG(1098) THEN
                        CALL rob.flow_off()
46                      SPEED 150 MMPS ALWAYS

48                      APPROS rob.pen_picpos1, 50
                        SPEED 60 MMPS ALWAYS
50                      MOVES rob.pen_picpos1
                        BREAK
52                      CALL rob.close_tool()
```

Program 1. *Pen picking program.*

For drawing process a certain requirements need to be met considering drawing coordinates. Since drawing is executed on plane surface robot needs X- and Y-coordinates for each drawing points and additional information when to raise pen from surface and when to place it back. This pen rising is carried out with Z-coordinate that can have value 0 or 1 depending if pen needs to be raised on that point or not. Drawing process operates with absolute positioning and executes drawing with same order they exist in received point list.

4.4.3 Error handling

In addition to robots internal E-stop functionalities a function was created to monitor the current state of the robot. This function was designed to loop in certain task and offer information to user about the nature of errors occurring. Another responsibility of error handler is to block incoming service calls during error states. This was done by terminating server process during the error state by error handler. After error handler detects that user has cleared all errors, the server process is started again. The basic principle of error handler functionality is introduced in figure 39.

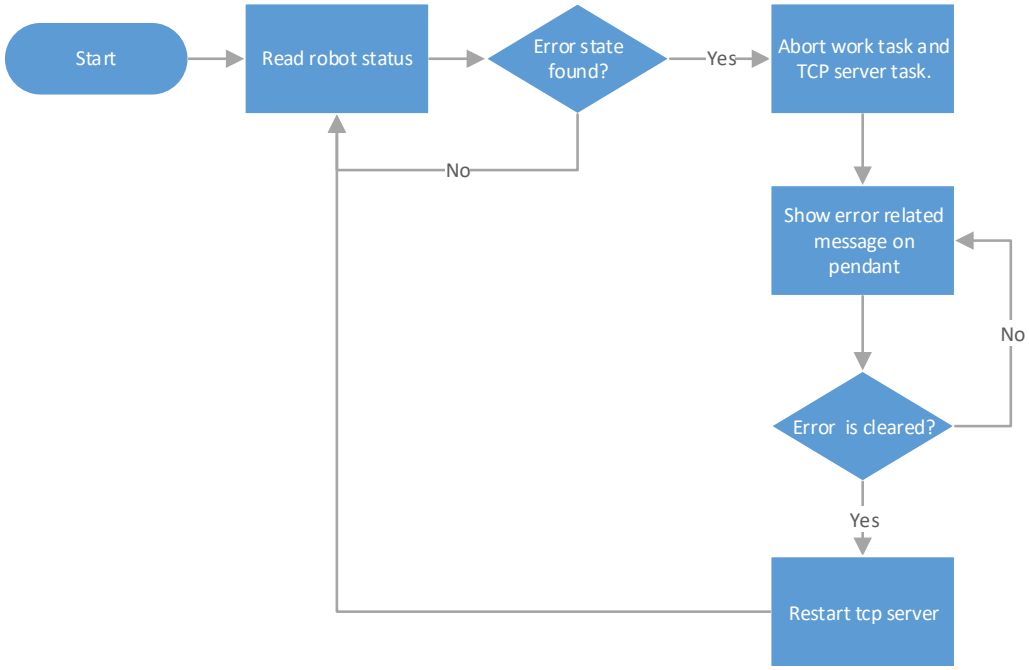


Figure 39. *Error handler task funktionality.*

This design stops current work task and leaves responsibility about robot repositioning to user. Design could be easily expanded to different types of error states. This implementation uses different kind of E-Stop signals to pick up error state. External safety

module is able this way to trigger different kind of error states using only the digital safety connections.

4.5 Free shape path point transferring

As designed in chapter 4.6 the draw adding process is implemented to interface of two TCP messages. First message “clear *1-9*” informs robot to clear one of the draw point lists. Second message includes the point information in following structure: “X: *** Y:*** Z:*** : *”. Message structure could be for example: “X:10.0 Y:20.0 Z:1 :5”. This message would add drawing point to location where X is 10. Y is 20. Z value 1 tells that point is not the end of a continuous line. So pen could not yet be raised from paper. Last number, five tells robot to which drawing list point is added.

Using these two messages an images could be added manually, but for sake of efficiency an external python script was implemented to convert SVG-files, generated by graphical drawing software, to list of points. Even free shape points could be calculated using robot, python environment has so many good libraries for SVG manipulation that there is no point to invent wheel again. After all, whole process of reading SVG-file, turning it into a list of points and sending those points to robot thru TCP interface was achieved with under 80 lines of code. The basic functionality of python script is introduced in figure 40.

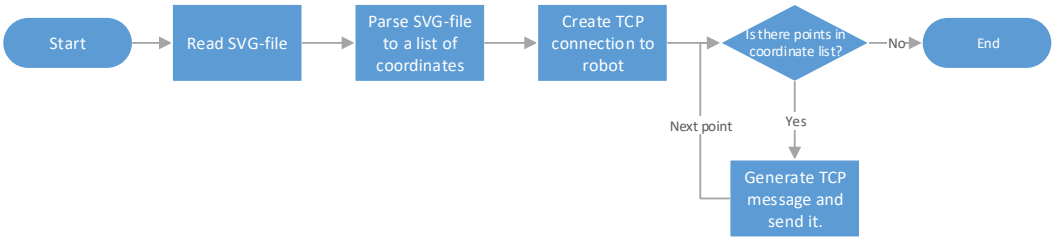


Figure 40. *Work flow of SVG parsing and point sending python script.*

This design could alter be expanded to support different types of image formats. At this point SVG is only supported. The whole script code is presented in appendix B.

5. CONCLUSIONS AND FUTURE WORK

The advances in communication and network technology has shifted automation system communication methods from using digital and analog I/O towards usage of more complex and efficient communicational technologies. Systematic standardization and system distribution led to development of field busses that were used mainly to link Programmable logic controllers to sensors, actuators, valves and electric motors. This offered more efficient data transformation between components, but also brought complexity to software configuration process. When computer networking technologies kept advancing, more and more efficient communication methods, like Ethernet, were invented and adapted to automation solutions. This development made possible to build more complex hardware modules that were capable to communicate with other devices in real time offering large set of complex data to other devices. Today many automation modules have high computation power and pre-built Ethernet connectivity, but the system designers need still to bring a lot of work to make these component interoperable.

The goal of this thesis was to find out how to integrate external modules to robot system maintaining modularity. The design and implementation phases showed that APIs are efficient way to integrate modules to robot system while maintaining modularity, scalability and reliability. To achieve these specifications, the API need to be well designed to support the robot's low-level functionality. APIs were designed and implemented for two different communicational level using two different communicational protocols. Low level APIs offered interface for RTU communication and movement point adding. These APIs were executed using TCP/IP message protocol. High level API between RTU and high level orchestrator were executed using HTTP based RESTful interface and Service Oriented Architecture. Inico S1000 RTU is a good example of well designed module that has the ability to create powerful API for system it is attached to.

Architectural design can be described as a understanding how a system components should be designed and organized in the scope of overall structure of the system. Architectural design is usually the first phase in the design process. Specially the software projects tend to have a person that is fully dedicated to system architecture design. This reflects the known importance of the system architecture design. It creates the important link between design and requirements because it identifies the structural components and the relationships between them. The architectural design process generates the architectural model. This model describes how the system is organized as a set of communicating components. When designing the architecture, the communication requirements are connection compatibility, real-time requirements and system responsibility analysis. Last one indicates to need to clarify if modules are stateless, or do they store system wise critical information.

The second goal of this thesis was to figure out how to create architectural design for robot system while focusing the communication architecture. When robot application includes supportive modules, which together with robot form multi-level communicational hierarchy, the architecture design need to be systematic and well documented. During the design phase the Unified Modelling Language appeared to be a useful tool in this type of applications. Mainly because of the systematic high to low level approach that offer models for structure, behaviour and interaction. UML offered create tools for hardware and software architecture design. During design phase the error handling occurred to be time consuming problem because communication between robot and RTU needed to be sequenced correctly to achieve desirable functionality. In case of emergency stop or power loss, system needs to recover successfully to the state it was before error situation. This was mainly achieved using centralized responsibility principle. All critical information about the system state was saved to robot memory and RTU acted as a stateless external module.

The third goal of this thesis was to find out an efficient way to control the robot using free shape paths. This created a problem considering the needed mathematical calculation. Even factory robots in general can handle this kind of calculations, in the sake of modularity the most reasonable choice was to take all calculation and 2D image processing to the environment that was more suitable to this kind of work. This would also make possible to parse different types of data-files to point based form that could be transferred to the robot.

The solution presented in this thesis separated the act of free shape calculation from robot to external module. The API for information transferring between this module and robot were also created to maintain modularity and scalability. Even though the De Casteljau Algorithm could have been implemented to robot, python programming language offers many open-source libraries for SVG-file parsing that speeded up development process significantly. Task that would have been complex, inflexible and hard to maintain when programmed to robot, was achieved under 80 lines of code. Maintaining modularity, scalability and flexibility.

As to the implemented system, the future improvements would consider the safety systems and draw point transfer. During the test phase showed that basic functionality of drawing and pen handling processes were quite reliable and offered good quality of products. As the external safety system was in design phase during the robot implementation, the totality of robot cell, safety circuit and conveyor system could not be tested. The image transfer API turned also to be reliable in practice and all nine images were transferred to robot system. When considering the future development of manufacturing line, the python script functionality would be improved to support free shape path uploading to multiple robots. When all the robots share the same image transfer interface, the image uploader script could handle the manufacturing line robots as a totality.

REFERENCES

[1] F. Lamb, Industrial Automation: Hands On, McGraw-Hill Education, 2013, .

[2] E.W. Kamen, Industrial Controls and Manufacturing, Academic Press, San Diego, 1999, .

[3] A.K. Gupta, S.K. Arora, J.R. Westcott, Industrial Automation and Robotics, Mercury Learning and Information, 2016, .

[4] M. Wilson, Implementation of Robot Systems : An Introduction to Robotics, Automation, and Successful Systems Integration in Manufacturing, Butterworth-Heinemann, Amsterdam, 2015, .

[5] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, Robotics: Modelling, Planning and Control, Springer London, 2010, .

[6] G.A. Bekey, Robotics: State Of The Art And Future Challenges, Imperial College Press, London, 2008, .

[7] J.L. Fuller, Robotics: introduction, programming, and projects, Merrill, 1991, .

[8] R.L. Hoekstra, Robotics and automated systems, South-Western Pub. Co, 1986, .

[9] Present and future robot control development—An industrial perspective, Industrial robotsRobot controlControl functionsControl applications, 2007, pp. 69-79.

[10] Implementation of Industrial Robot for Painting Applications, in: International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012), ABB Industrial Robot-Painting RobotIRC5 ControllerFlexpendantRobotstudioSolidWorks, 2012, pp. 1329-1335.

[11] Seamless human robot collaborative assembly – An automotive case study, Human robot collaborationInteractionAugmented realityWearable devicesSafetyIntegration and communication, 2018, pp. 194-211.

[12] Recent progress on programming methods for industrial robots, Industrial robotSMEsOffline programmingOnline programmingAugmented Reality, 2012, pp. 87-94.

[13] D. Massa, M. Callegari, C. Cristalli, Manual guidance for industrial robot programming, The Industrial Robot, Vol. 42, No. 5, 2015, pp. 457-465.

[14] Off-line programming of Spot-weld Robot for Car-body in White Based on Robcad, in: 2007 International Conference on Mechatronics and Automation, 2007, pp. 763-768.

[15] An industrial application of control of dynamic behavior of robots-a walk-through programmed welding robot, in: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), 2000, pp. 2352-2357 vol.3.

[16] Intuitive robot programming through environment perception, augmented reality simulation and automated program verification, in: 7th CIRP Conference on Assembly Technologies and Systems (CATS 2018), Industrial RoboticsRobot ProgrammingAugmented Reality, 2018, pp. 161-166.

[17] The Development of Motion Capture System Based on Kinect Sensor and Bluetooth-Gloves, in: Proceedings of the 3rd International Conference on Dynamics and Vibroacoustics of Machines (DVM2016) June 29–July 01, 2016 Samara, Russia, Motion capture systemanthropomorphic robotAR-600EKinectBluetoothLabVIEWremote control calculations, 2017, pp. 506-513.

[18] J. Aleotti, S. Caselli, Grasp programming by demonstration in virtual reality with automatic environment reconstruction, Virtual Reality, Vol. 16, No. 2, 2012, pp. 87-104.

[19] R. Dillmann, O. Rogalla, M. Ehrenmann, R. ZÃ¶liner, M. Bordegoni, Learning Robot Behaviour and Skills Based on Human Demonstration and Advice: The Machine Learning Paradigm, Robotics Research, Springer London, London, pp. 229-238.

[20] J.N. Pires, G. Veiga, R. Araújo, Programming-by-demonstration in the coworker scenario for SMEs, The Industrial Robot, Vol. 36, No. 1, 2009, pp. 73-83.

[21] C. Machover, R.E. Blauth, C. Corp, The CAD/CAM handbook, Computervision Corp, 1980, .

[22] A. Tizzard, An Introduction to Computer-aided Engineering, McGraw-Hill, 1994, .

[23] D.D. Voisinet, Introduction to Computer-Aided Drafting, Gregg Division, McGraw-Hill, 1983, .

[24] G.R. Bertoline, Fundamentals of CAD, Delmar Publishers, 1988, .

[25] Computer-aided design and Computer-aided engineering, EDP Sciences, EngineeringComputing costsSystem effectivenessDesign engineeringCAD/CAMCost reductionComputing timeNew technologyComputer simulationComputer aided engineering--CAEComputer aided design, Les Ulis, France Les Ulis, Les Ulis, 2018, pp. n/a.

[26] W.H. Phillips, Additive Manufacturing : Opportunities, Challenges, Implications, Nova Science Publishers, Inc, New York, 2016, .

[27] I. Gibson, D.W. Rosen, B. Stucker, Introduction and Basic Principles, in: I. Gibson, D.W. Rosen, B. Stucker (ed.), Additive Manufacturing Technologies: Rapid Prototyping to Direct Digital Manufacturing, Springer US, Boston, MA, 2010, pp. 20-35.

[28] C. Coward, 3D Printing, in: Alpha Books, 2015, .

[29] J. Horvath, The 3D Printing Process, in: Mastering 3D Printing, Apress, 2014, pp. 30-10.

[30] W. Goralski, Chapter 1. Protocols and Layers, in: The Illustrated Network : How TCP/IP Works in a Modern Network, Morgan Kaufmann, Amsterdam, 2009, pp. 14.

[31] D. Kabelová, Introduction to Network Protocols, in: Understanding TCP/IP : A Clear and Comprehensive Guide, Packt Publishing, Limited, Olton, 2006, pp. 27.

[32] T. Herbert, Linux TCP/IP Stack : Networking for Embedded Systems, in: Charles River Media, Herndon, 2004, pp. 52.

[33] The OSI Network Model Explained, OSI (Computer network standard)Computer networksInternetData packetingComputer simulationCiphers, 2012, pp. 3-4.

[34] M. Rosen, B. Lublinsky, K.T. Smith, Applied SOA : Service-Oriented Architecture and Design Strategies, in: John Wiley & Sons, Incorporated, Hoboken, 2003, pp. 28.

[35] S. Anandamurugan, T. Priyaa, Service Oriented Architecture, in: Nova Science Publishers, Inc, Hauppauge, New York, 2014, pp. 67-68.

[36] H. Subramanian, P. Raj, Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs, in: Packt Publishing, 2019, pp. 11-12.

[37] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, in: Pearson Education, 2005, pp. 43-44.

[38] J. Sandoval, RESTful Java Web Services : Master Core REST Concepts and Create RESTful Web Services in Java, in: Packt Publishing, Limited, Olton, 2009, pp. 24.

[39] T. Erl, B. Carlyle, C. Pautasso, SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST, in: Prentice Hall, 2012, pp. 52.

[40] M. Masse, REST API Design Rulebook, in: O'Reilly Media, 2011, pp. 5-6.

[41] M. Laaker, Sams Teach Yourself SVG in 24 Hours, in: Sams, 2002, pp. 7.

[42] A. Libby, Beginning SVG: A Practical Introduction to SVG using Real-World Examples, in: Apress, 2018, pp. 3-4.

[43] On de Casteljau-type algorithms for rational Bézier curves, Rational Bernstein functionsDe Casteljau-type algorithmRational Bézier curvesLupaş -analogue of Bernstein operatorDegree elevation, 2015, pp. 244-250.

[44] G. Farin, Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide, Elsevier Science, 2014, .

[45] M. Kim, J. Hoschek, G.E. Farin, Handbook of Computer Aided Geometric Design, North Holland, Amsterdam, 2002, .

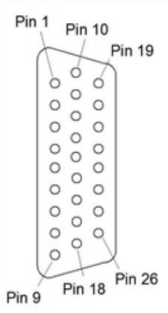
[46] FASTory Simulation, Tampere University of Technology, web page. Available (accessed 7/2019): http://escop.rd.tut.fi:3000/fmw.

[47] Omron, Viper 650/850 Robot with eMB-60R User's Guide, Omron Adept Technologies, web page. Available (accessed 8/2019): https://assets.omron.com/m/6024ba162c07ee40/original/Viper-650-850-Robot-with-eMB-60R-User-Guide.pdf.

[48] S1000 User Manual, Inico Technologies, web page. Available (accessed 10/2019): https://www.inicotech.com/doc/S1000%20User%20Manual.pdf.

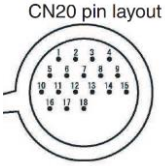
APPENDIX A: TABLE OF ROBOT CONNECTIONS

XIO Connector		CN20 connector		CN21 connector		eV+ Signal Number	Pin Locations
pin	usage	pin	usage	pin	usage		
1	GND	1	GND for sensors	1	GND		
2	24 VCD	2	24 VCD for sensors	2	24 VCD		
3	Common1	8	common (GND)	8	common GND		
4	input 1.1	3	for sensors	3	pen presence sensor	1097	
5	input 2.1	4	for sensors	4	tool opening sensor	1098	
6	input 3.1	5	for sensors	5	vertical presence sensor	1099	
7	input 4.1						
8	input 5.1						
9	input 6.1						
10	GND	12	Solenoid ground				
11	24 VDC						
12	Common2						
13	input 1.2						
14	input 2.2						
15	input 3.2						
16	input 4.2						
17	input 5.2						
18	input 6.2						
19	output 1	13	Solenoid 1A			0097	
20	output 2	14	Solenoid 1B			0098	
21	output 3	15	Solenoid 2A			0099	
22	output 4	16	Solenoid 2B			0100	
23	output 5	17	Solenoid 3A			0101	
24	output 6	18	Solenoid 3B			0102	
25	output 7						
26	output 8						



XIO 26-pin female connector on eMB-60R Interface Panel

CN20 pin layout



APPENDIX B: SVG PARSING AND TRANSFERRING SCRIPT.

```
### Robot image transfer script
#
# This script was created for converting SVG-file image information to point-based structure
# and to upload the points to industrial robot using specified TCP plain text interface.
#
# Script supports SVG Path, Line, Polyline, and Polygon elements,
# these constraints originate to used svgpathtools library.
#
#Expected comman line arguments:
# sys.argv[1] = name/path to uploaded svg image.
#
# sys.argv[2] = number 1-9 reflecting image slot where image is transfered in robto system.
#
'''

import sys
import svg.path
import time

from svgpathtools import svg2paths

import socket

coordList = []              #object where point information will be stored
TCP_IP = '192.168.5.100'    #robot system IP adress
TCP_PORT = 3000             #robot system TCP port number
BUFFER_SIZE = 1024
MESSAGE = "Hello, World!"

print ("process start")
print("please use form -python svgC.py fileName.svg PictureNumber")
paths, attributes = svg2paths(sys.argv[1])   #convert SVG image elements to paths object

for path in paths:              #go through all elements in paths object
  print("path:" + str(path))
  for j, seg in enumerate(path):
    i = 0
    z = 1
    while i < 6: # divide path into 6 points
      print("SEG")
      print (j)
      print(len(path))

      if i == 5 and j == len(path)-1:
        #change z value to 0 if point is at the end of the path line.
        #This tells robot to raise the tool from surface.
        z = 0
      #create tuplex ingluding x and y coordinates and z inform value (z=0 or z=1)
      tuplex = (round(seg.point(i/5).real,2),round(seg.point(i/5).imag,2),z)
      coordList.append(tuplex)
      print(i)
      print(tuplex)
      i += 1

#when list of points is created to coordList object
#TCP-socet is opened
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))

MESSAGE = "clear" + sys.argv[2]
s.send(MESSAGE.encode('utf-8'))

#send points to robot using TCP socket
#example of a  sended message is "X:5.5 Y:10.2 Z:1:5
#where 5.5=x-coordinate 10.2 is y-coordinate 1 is z inform value
#and 5 is image slot iamge is transfered to.

for k, coord in enumerate(coordList):
  print(str(round(k/len(coordList)*100,2)) + "%")

  MESSAGE = "X:"+ str(coord[0]) + " Y:" + str(coord[1])+ " Z:" + str(coord[2]) + ":" + sys.argv[2]
  print(MESSAGE)
  s.send(MESSAGE.encode('utf-8'))
  data = s.recv(BUFFER_SIZE)
  print(data)

s.close()
print("Succesfully updated " + sys.argv[1] +" to pictureslot "+ sys.argv[2])
```

APPENDIX C: ROBOT PROGRAMS

```
.PROGRAM rob.draw(runpoints[,])

        GLOBAL LOC drawframe
        GLOBAL REAL rob.run
        LOCAL REAL last_z
        ATTACH (0)

        robrun = TRUE

        CALL rob.flow_on()
        SPEED 280 MMPS ALWAYS
        last_z = 0


        FOR indx = 0 TO LAST(runpoints[,])

            IF NOT (runpoints[indx,1] == -1) THEN

                IF last_z == 0 THEN
                    SET nextdrawpos = drawframe:TRANS(runpoints[indx,1],runpoints[indx,2],-5,0,-90,0)
                    MOVES nextdrawpos
                    BREAK
                    SET nextdrawpos = drawframe:TRANS(runpoints[indx,1],runpoints[indx,2],0,0,-90,0)
                    MOVES nextdrawpos
                    BREAK

                ELSE
                    SET nextdrawpos = drawframe:TRANS(runpoints[indx,1],runpoints[indx,2],0,0,-90,0)
                    MOVES nextdrawpos
                END

                IF runpoints[indx,3] == 0 THEN
                    BREAK
                    MOVES nextdrawpos:TRANS(-5,0,0)
                    BREAK
                END
                last_z = runpoints[indx,3]
            END
        END

        MOVES nextdrawpos:TRANS(-10,0,0)
        BREAK
        MOVE rob.home
        BREAK
        CALL rob.flow_off()
        rob.run = FALSE
        SIGNAL (-103)

.END
```

```
.PROGRAM rob.draw_conf()

        exit = FALSE
        IF NOT SIG(1097) THEN
            PDNT.NOTIFY "Info", "Pen needs to be picked before configuration."
        END

        IF SIG(1097) THEN

            SPEED 300 MMPS ALWAYS
            MOVES loc.draworigin:TRANS(-5,0,0) ;Initiate motion to next location
            CALL rob.flow_on() ;
            SPEED 10 MMPS ALWAYS
            MOVES loc.draworigin:TRANS(30,0,0)

            DO ;Loop continuously...
                IF NOT SIG(1099) THEN ;If input signal 1023 becomes set,
                    BRAKE ;stop the motion immediately
                    EXIT ;and continue elsewhere
                END
            UNTIL exit

            HERE loc.draworigin
            WAIT.EVENT , 0.5
            MOVES loc.draworigin:TRANS(-3.45,0,0)
            BREAK
            HERE loc.draworigin
            WAIT.EVENT , 0.5
            MOVES loc.draworigin:TRANS(-5,0,0)

        ;configure x direction
            SPEED 300 MMPS ALWAYS
            MOVES loc.drawx:TRANS(-5,0,0) ;Initiate motion to next location
            CALL rob.flow_on() ;
            SPEED 10 MMPS ALWAYS
            MOVES loc.drawx:TRANS(30,0,0)

            DO ;Loop continuously...
                IF NOT SIG(1099) THEN ;If input signal 1023 becomes set,
                    BRAKE ;stop the motion immediately
                    EXIT ;and continue elsewhere
                END
            UNTIL exit

            HERE loc.drawx
            WAIT.EVENT , 0.5
            MOVES loc.drawx:TRANS(-3.45,0,0)
            BREAK
            HERE loc.drawx
            WAIT.EVENT , 0.5
            MOVES loc.drawx:TRANS(-5,0,0)
```

```
53
54
55
56          ;configure y direction
57              SPEED 300 MMPS ALWAYS
58              MOVES loc.drawy:TRANS(-5,0,0) ;Initiate motion to next location
59              CALL rob.flow_on() ;
60              SPEED 10 MMPS ALWAYS
61              MOVES loc.drawy:TRANS(30,0,0)
62
63              DO ;Loop continuously...
64                  IF NOT SIG(1099) THEN ;If input signal 1023 becomes set,
65                      BRAKE ;stop the motion immediately
66                      EXIT ;and continue elsewhere
67                  END
68              UNTIL exit
69
70
71              HERE loc.drawy
72              WAIT.EVENT , 0.5
73              MOVES loc.drawy:TRANS(-3.45,0,0)
74              BREAK
75              HERE loc.drawy
76              WAIT.EVENT , 0.5
77
78              SPEED 250 MMPS ALWAYS
79              MOVES loc.drawy:TRANS(-5,0,0)
80
81          END
82
83          SPEED 250 MMPS ALWAYS
84          MOVES rob.home
85
86          SET drawframe = FRAME(loc.draworigin,loc.drawx,loc.drawy,loc.draworigin)
87
88          rob.run = FALSE
89          SIGNAL (-103)
90
91  .END
92
```

```
1   .PROGRAM rob.drop_pen()
2
3           IF pen_on_board == 1 THEN
4               CALL rob.drop_pen1()
5           END
6           IF pen_on_board == 2 THEN
7               CALL rob.drop_pen2()
8           END
9           IF pen_on_board == 3 THEN
10              CALL rob.drop_pen3()
11          END
12
13  .END
14
```

```
.PROGRAM rob.drop_pen1()
; INPUTS:
;   SIG 1097   - TRUE = pen present in gripper
;   SIG 1098   -  TRUE = tool open
        GLOBAL LOC rob.pen_picpos1
        GLOBAL REAL rob.run

        CALL rob.flow_off()
        SPEED 400 MMPS ALWAYS
        MOVES rob.pen_picpos1:TRANS(-100,0,0)
        BREAK
        SPEED 60 MMPS ALWAYS
        MOVES rob.pen_picpos1:TRANS(1,0,0)
        BREAK
        WAIT.EVENT , 1
        CALL rob.open_tool()
        WAIT.EVENT , 1

        IF SIG(1098) THEN
            pen_on_board = 0
            MOVES rob.pen_picpos1:TRANS(-0,0,-50)
            BREAK
            MOVES rob.safe
            BREAK

        END
        IF NOT SIG(1098) THEN
            MOVES rob.pen_picpos1:TRANS(-100,0,0)
            BREAK
            SPEED 150 MMPS ALWAYS
            MOVES rob.home
            BREAK


        END

.END
```

```
.PROGRAM rob.pick_pen1()
; INPUTS:
;   SIG 1097   - TRUE = pen present in gripper
;   SIG 1098   -  TRUE = tool open
;
        ;Pen Colors
        ;BLUE = 1
        ;GREEN = 2
        ;RED = 3
; DATA STRUCT

        GLOBAL LOC rob.pen_picpos1
        GLOBAL REAL rob.run
        SPEED 60 MMPS ALWAYS
        ATTACH (0)
        rob.run = TRUE

        ;if pen present on gripper
        IF SIG(1097) THEN

            CALL rob.drop_pen()
        END

        ;make sure that there is no pen on gripper
        IF NOT SIG(1097) THEN
            CALL rob.open_tool()
            WAIT.EVENT , 1

            IF NOT SIG(1098) THEN
                PDNT.NOTIFY "Pneumatic problem", "can't open gripper"
            END

            IF SIG(1098) THEN
                CALL rob.flow_off()
                SPEED 200 MMPS ALWAYS

                APPROS rob.pen_picpos1, 50
                SPEED 60 MMPS ALWAYS
                MOVES rob.pen_picpos1
                BREAK
                CALL rob.close_tool()

                WAIT.EVENT , 1

                IF NOT SIG(1098) THEN
                    pen_on_board = 1
                    SPEED 100 MMPS ALWAYS
                    MOVES rob.pen_picpos1:TRANS(-100,0,0)
                    BREAK
                END

                SPEED 200 MMPS ALWAYS
                MOVES rob.home
                BREAK
            END
        END

        rob.run = FALSE
        SIGNAL (-103)

.END
```